UNIVERSIDAD CARLOS III DE MADRID
ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA TÉCNICA DE TELECOMUNICACIÓN
SONIDO E IMAGEN

PROYECTO FIN DE CARRERA

ESTUDIO DE LA EFICIENCIA DE PROTOCOLOS Y ALGORITMOS DE SEGURIDAD EN ANDROID

Autor: Elma de la Fuente

Director: Florina Almenárez Mendoza

Tutor: Patricia Arias Cabarcos

Título: ESTUDIO DE LA EFICIENCIA DE PROTOCOLOS Y ALGORITMOS DE SEGURIDAD EN ANDROID

Autor: ELMA DE LA FUENTE

Directores: FLORINA ALMENÁREZ MENDOZA Y PATRICIA ARIAS CABARCOS

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día 14 de octubre de 2015 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

Resumen

Resulta innegable que Internet y los dispositivos móviles han modificado nuestro estilo de vida hasta niveles inconcebibles desde hace tan solo unos años. Los usuarios de teléfonos móviles han ido incrementando considerablemente en la última década y con ello el uso de Internet desde dichos dispositivos. Mientras que la navegación y el intercambio de información en Internet suponen un gran beneficio y facilitan la comunicación digital entre individuos, es importante conocer cuáles son las principales amenazas asociadas y adoptar medidas de protección adecuadas para eliminar, o al menos reducir, nuestro nivel de riesgo.

El presente proyecto realiza un análisis detallado de los algoritmos criptográficos y protocolos de seguridad para uno de los principales sistemas operativos de dispositivos móviles utilizados en la actualidad, Android, en concreto hasta la versión 2.2, con el objetivo de determinar el tipo de cifrado que puede aplicarse en conexiones seguras en Internet, protección de aplicaciones o en almacenamiento de datos en local.

Palabras clave:

Protocolos de seguridad, Android, cifrado clave simétrica, cifrado clave asimétrica, firma digital, conexión TLS

Abstract

It is irrefutable that the Internet and mobile devices have changed our life's style to levels unimaginable from a few years ago. Mobile phone users have increased considerably over the last decade and with them the use of the Internet from such devices. While navigation and the exchange of information on the Internet are a great benefit and make easier digital communication between people, it is also important to be aware what are the main associated threads and take adequate security measures to eliminate or at least reduce our level of risk.

This project makes a detailed cryptographic algorithms and security protocols analysis of a major mobile operating system used nowadays, Android, specifically to the version 2.2, in order to determinate the type of encryption that can be applied in the Internet security connections, applications protection or through locally data storage.

Key words:

Security protocols, Android, symmetric key cipher, asymmetric key cipher, digital sign, TLS connection

Índice general

Índice de Figuras

ÍNDICE DE TABLAS

Capítulo 1
Introducción y objetivos

Este capítulo sirve de presentación general del proyecto fin de carrera. Primero se expondrá una breve introducción para situar al lector en el marco tecnológico en el que se encuentra este proyecto. Posteriormente se hablará sobre las motivaciones que han dado lugar al mismo y los objetivos marcados para su desarrollo. Como punto final, se describirá cómo se ha estructurado esta memoria.

1.1. Introducción

La popularización del teléfono móvil no deja de ser un fenómeno relativamente reciente. Hemos pasado de un dispositivo de comunicación entre dos personas vía voz a un dispositivo dotado de mayor complejidad, en el que cada vez va incorporando más funcionalidades.

Estas nuevas generaciones de terminales nacen de la necesidad de aumentar la capacidad de transmisión de datos para poder ofrecer servicios como la conexión a internet desde el móvil, videoconferencia y descarga de archivos. Chequear los saldos bancarios o realizar compras a través del teléfono, por ejemplo, son prácticas que se están extendiendo. El Smartphone se ha convertido en un dispositivo vital en nuestras vidas. Hoy en día, un mayor número de consumidores utiliza su Smartphone para tareas que antes solo estaban disponibles a través de PC.

La complejidad de los ordenadores incorporada a estos dispositivos supone grandes beneficios para los usuarios, pero también conlleva riesgos relacionados con la seguridad, propiciados por el volumen de información personal que se almacena en ellos y que puede suponer un objetivo interesante para cibercriminales.

Las amenazas están presentes prácticamente en cualquier sistema operativo y arquitectura.

Por eso, este proyecto surge de la necesidad de evaluar los algoritmos, protocolos y técnicas disponibles en uno de los sistemas operativos móviles más extendidos, Android.

1.2. Objetivos

El objetivo de este proyecto es estudiar la eficiencia de algoritmos y protocolos de seguridad en dispositivos móviles con sistema operativo Android.

El alcance de este estudio comprende los siguientes puntos:

- Estudio de eficiencia de algoritmos de cifrado en clave simétrica y asimétrica en términos de coste temporal.
- Estudio de eficiencia de algoritmos de firma digital en términos de coste temporal.
- Estudio de eficiencia de las conexiones TLS/SSL soportados por el navegador
- Programación e implementación de una aplicación que realice la medida de eficiencia de algoritmos y protocolos de seguridad estudiados (cifrado, descifrado, firma y verificación de documentos)

1.3. Fases de desarrollo

Las fases de desarrollo del proyecto comprenden:

1. Estudio previo
    a. Estudio de Android, de su API (*Application Program Interface*) de seguridad, y algoritmos de la librería OpenSSL
    b. Estudio de los algoritmos de cifrado simétricos
    c. Estudio de algoritmos de cifrado asimétrico
    d. Estudio de eficiencia de conexiones TLS/SSL
2. Implementación
    a. Desarrollo de la aplicación para la medición de los algoritmos de seguridad
    b. Configuración de los parámetros de los algoritmos
    c. Extracción de resultados
3. Análisis de los resultados y documentación: redacción de la memoria

1.4. Estructura de la memoria

La estructura de la memoria es la siguiente:

- Capítulo 2: se define el estado del arte de los algoritmos de seguridad, introduciendo las nociones básicas de criptografía;
- Capítulo 3: se define el estado del arte del sistema operativo para dispositivos móviles Android;
- Capítulo 4: se estudia la eficiencia de los algoritmos de cifrado simétrico y asimétrico en cuanto a seguridad y rapidez se refiere;
- Capítulo 5: se presenta el estudio realizado sobre la eficiencia de algoritmos de firma digital;
- Capítulo 6: se presenta el estudio de la eficiencia de conexiones TLS/SSL;
- Capítulo 7: se resumen las principales conclusiones del proyecto y se enumeran una serie de posibles líneas de trabajo futuro;
- Capítulo 8: se presenta el presupuesto asociado al desarrollo del proyecto.

Capítulo 2
Seguridad y Criptografía

El uso generalizado de redes de comunicación en el tratamiento y transmisión de la información, así como el aumento constante del número de usuarios de estos sistemas, han motivado la necesidad de mejorar la seguridad en las comunicaciones. Son muchas y variadas las situaciones donde cabe garantizar la privacidad, la integridad o la autentificación de la información transmitida. Tales necesidades se han podido satisfacer mediante el uso de distintos protocolos criptográficos, en los que se combinan a menudo criptosistemas de clave compartida con criptosistemas de clave privada.

2.1. Principios de la Criptografía

La *Criptología*, que en un sentido etimológico proviene del griego *Kripto* (oculto) y *Logos* (tratado) (RAE n.d.), es el estudio y práctica de los sistemas de cifrado destinados a garantizar la privacidad en el intercambio de información, es decir, hacer inteligible un mensaje para cualquier elemento no autorizado en la comunicación.

La criptología puede dividirse en dos disciplinas fundamentales: la *Criptografía* y el *Criptoanálisis*.

La *Criptografía* hace uso de métodos y técnicas con el objeto principal de enviar un mensaje desde un emisor a un receptor sin que nadie que intercepte el mensaje en el camino pueda interpretarlo.

El *Criptoanálisis* se basa en los mecanismos utilizados para decodificar estos mensajes, es decir, busca romper los procedimientos de cifrado y así conseguir el mensaje original. (Lucena López 2011)

Hoy en día el intercambio de información valiosa por Internet es una práctica común. En general, toda comunicación puede quedar expuesta a la posibilidad de un ataque de un adversario, por lo que proteger los datos intercambiados se ha vuelto una tarea prioritaria en todo ámbito. La criptografía busca resolver estos problemas, implícitos con el manejo de la información, a través de una serie de controles, políticas o procedimientos destinados a preservar (Sánchez 2005):

> *Confidencialidad / Privacidad*: la información solo debe ser accesible a aquellos autorizados a obtenerla. Se debe impedir la revelación no autorizada de ésta.
>
> *Integridad*: la información solo debe ser alterada por aquellos autorizados, pudiéndose comprobar que no haya sido manipulada según el original. Se debe prevenir la modificación no autorizada de ésta.
>
> *Autentificación*: verifica que el origen de un documento pertenece a quien dice ser.
>
> *No repudio*: impide que un individuo niegue haber realizado una transacción, ya sea emitiendo y recibiendo un mensaje, cuando efectivamente lo ha hecho. Es necesario que una tercera parte de confianza resuelva dicha disputa.

Si se cumplen estas propiedades se considera que los datos están protegidos y seguros.

El siguiente ejemplo ilustra el escenario básico de un sistema criptográfico (Lucena López 2011):

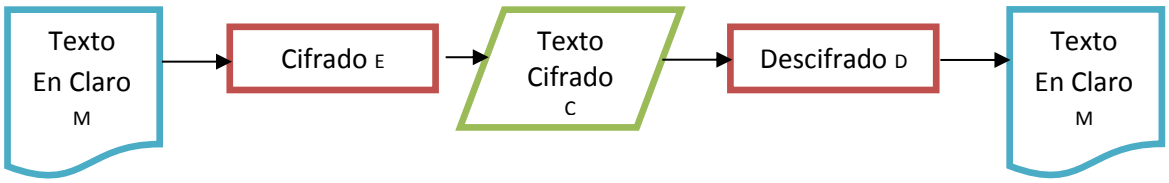


Figura 1. Esquema básico sistema criptográfico

- El texto en claro, M, es el mensaje a transmitir de forma confidencial.
- El proceso de *cifrado*, E, transforma el texto plano M en texto cifrado o criptogramas, C, mediante el uso de un conjunto de claves K.
- El *descifrado*, D, es el proceso inverso, capaz de transformar el texto cifrado en el texto plano original.

  Todo criptosistema ha de cumplir la siguiente condición:

$$D_k(E_k(m)) = m$$

  Por la cual dado un mensaje cifrado, tras la correcta aplicación del descifrado con su clave correspondiente, es posible hallar el mensaje original.

2.2. Clasificación de la Criptografía

En la criptografía moderna se pueden clasificar los diferentes sistemas de cifrado existentes en dos grandes grupos basándose en el tipo de clave utilizada: la criptografía de clave secreta o simétrica, y la criptografía de clave pública o asimétrica.

No obstante, dentro de la criptografía simétrica, se puede hacer una diferenciación entre los cifradores en bloque y los cifradores de flujo atendiendo al tratamiento de la información que éstos realizan.

2.2.1. Criptografía Simétrica

Son algoritmos que utilizan la misma clave secreta para el cifrado y descifrado del mensaje. Ambos extremos de la comunicación deben ponerse de acuerdo en usar una única clave, intercambiada a través de un canal seguro, para cumplir con el proceso. Por tanto, toda la seguridad de estos algoritmos está basada en la privacidad de esta clave secreta. (Álvarez n.d.)

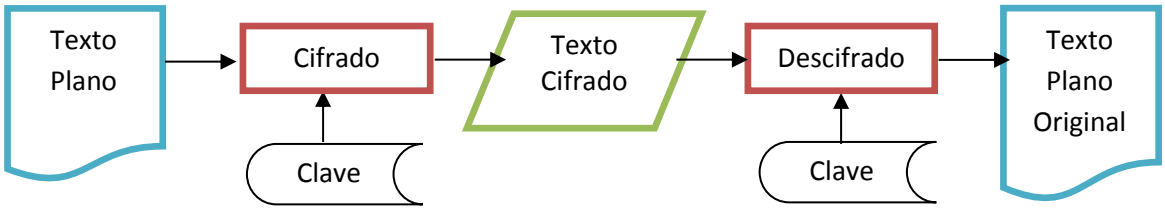


Figura 2. Esquema sistema criptográfico simétrico

Para que sea fiable, debe cumplir las siguientes condiciones:

- Conocido del criptograma no se puede obtener de él el texto plano ni la clave
- Conocidos el texto plano y el cifrado, debe resultar más costoso descifrar la clave que el valor posible de la información obtenida.

Su principal desventaja reside en la distribución de la clave. Ningún canal de comunicación es lo bastante seguro como para garantizar que nadie intercepte el mensaje por el que se envía la clave.

Otro problema es el número de claves necesarias. Para que un número *n* de personas se comuniquen entre sí, son necesarias *n-1* claves para cada una, lo que supone $n(n-1/2)$ claves en total.

Como ya se había adelantado, se puede hacer una clasificación de los algoritmos de cifrado simétricos en dos categorías: los *cifradores de bloque* que trabajan sobre un bloque de bits de un tamaño tal que impida un análisis trivial pero que resulte cómodamente manejable; y los que trabajan sobre un único elemento de flujo de comunicación (un bit, un byte o un carácter) que se denominan *cifradores de flujo*.

Los cifradores en bloque son generalmente más eficientes en software ya que permiten trabajar con bloques de tamaño compatible con los registros del procesador. Los cifradores en flujo presentan ventajas en situaciones de tiempo real, en las que no resulta viable esperar a tener un bloque completo antes de enviar la información.

A continuación se procede a describir ambas tipologías.

2.2.1.1. Criptografía Simétrica por Bloques

Este tipo de criptografía está basada en el diseño propuesto por Horst Feistel en los años 70, apoyado sobre conceptos de:

> *Confusión:* o sustituciones simples. Se trata de ocultar la relación existente entre el texto plano, el cifrado, y la clave; y
>
> *Difusión*: o permutaciones, que tratan de repartir la influencia de cada bit del mensaje original lo más posible en el mensaje cifrado. (Lucena López 2011)

De esta forma, se divide el mensaje en bloques de tamaño fijo y se aplica la función de cifrado a cada uno de ellos aplicando las operaciones mencionadas. Algunos de los algoritmos criptográficos que utilizan esta filosofía son DES y TripleDES, IDEA o AES.

Si utilizamos algoritmos con métodos de funcionamiento con tamaño de bloque fijo, habrá ocasiones que la longitud de la cadena a cifrar no sea múltiplo exacto del tamaño del bloque definido por el algoritmo. La solución en estos casos se realiza a través de un mecanismo conocido como *padding*, de forma que se añade información extra al final del bloque, rellenando con ceros o cualquier otro patrón, hasta completarlo. Para determinar en qué momento ha comenzado el relleno, se debe expresar la información extra añadida en el último byte. Si el tamaño original del texto era múltiplo del bloque, se tendrá que alargar con un nuevo bloque entero del patrón seleccionado.

La forma en la que se mezcla el texto plano con la clave define los diferentes modos de funcionamiento del algoritmo:

- Modo ECB: *Electronic Code Book*

  Subdivide el texto plano en bloques de un tamaño determinado y se cifran todos ellos empleando la misma clave, sin ningún tipo de realimentación.

  Al cifrarse los bloques independientemente de su orden, resulta adecuado en información que requiera un acceso aleatorio. Es resistente a errores, de forma que si uno de los bloques sufre una alteración, el resto quedará intacto.

  Por el contrario, si el texto plano presenta patrones repetitivos, el texto cifrado también los presentará, estando expuesto a un ataque estadístico por el que se puede extraer la información confidencial por la presentación de dicha redundancia. Así mismo, debido a su diseño independiente, también se puede presentar el riesgo de sustitución de bloques por algún atacante que escuche la conversación.

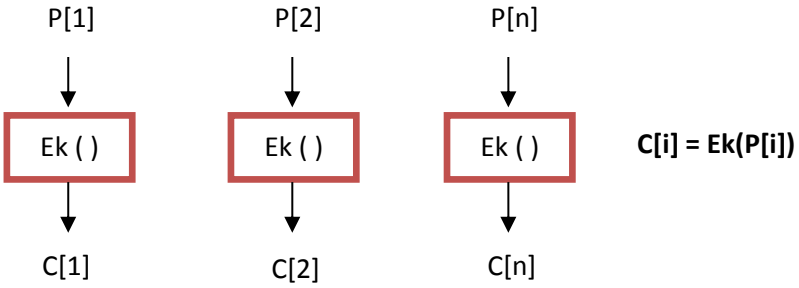


Figura 3. Modo ECB

- Modo CBC: *Cipher Block Chaining*

  En este caso se incorpora un mecanismo de retroalimentación de los bloques, de forma que la codificación del bloque actual está condicionada por la codificación del bloque anterior. Esto se consigue mediante una operación XOR entre el bloque a codificar y el criptograma del cifrado anterior.

  Al presentar esa dependencia con respecto a bloques anteriores, el receptor detectará cualquier ataque de supresión y/o inserción de bloques.

  Para comenzar la codificación, se utiliza un vector de inicialización, de carácter aleatorio, como bloque inicial de la transmisión. Este vector será descartado en destino, y garantizará que los mensajes sean codificados de forma diferente ya que de otra forma, mensajes iguales obtendrían las mismas salidas.

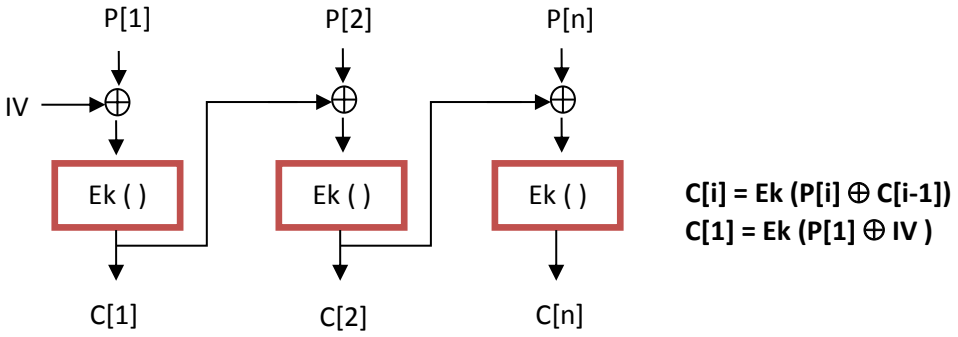
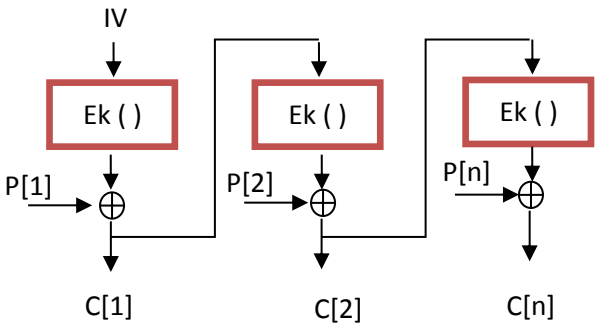


Figura 4. Modo CBC

- Modo CFB: *Cipher Feed Back*

Cada bloque es cifrado mediante la operación XOR con una combinación de la clave y el bloque cifrado anterior. Es necesario emplear un vector de inicialización de carácter aleatorio del tamaño del bloque para codificar el primero.

Este modo permite codificar la información en unidades inferiores al tamaño del bloque del algoritmo, de forma que se aprovecha totalmente la capacidad de transmisión del canal de comunicaciones, manteniendo un nivel de seguridad adecuado.



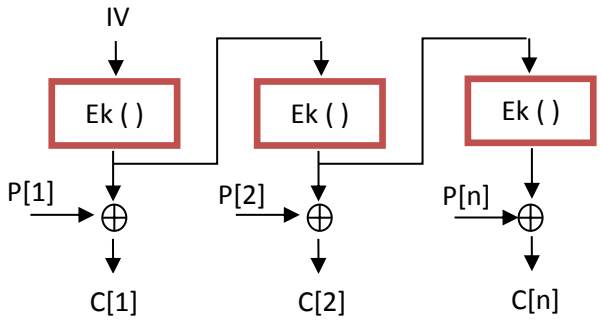
C[i] = P[i] ⊕ Ek (C[i-1])
C[1] = P[1] ⊕ Ek (IV)

Figura 5. Modo CFB

- Modo OFB: *Output Feed Back*

A partir de una clave K y de un vector de inicialización, genera de forma totalmente independiente al mensaje una secuencia pseudoaleatoria (oi) que se aplica en XOR con el texto en claro para generar el texto cifrado.

Este funcionamiento puede ser empleado como generador de secuencia de clave, utilizado en algoritmos de cifrado por flujo.



C[i] = P[i] ⊕ O[i]
O[i] = Ek (O[i-1])
O[1] = Ek (IV)

Figura 6. Modo OFB

2.2.1.2. Criptografía Simétrica de Flujo

Este tipo de criptografía se basa en cifrar bit a bit, o byte a byte, por lo que están diseñados idealmente para aquellas comunicaciones que se estén generando en tiempo real, como por ejemplo en telefonía móvil digital. Los algoritmos RC4, A5 y Seal son ejemplos de este tipo de cifradores.

Al cifrar unidades mínimas de contenido, sin que éstas dependan de algo más (no existe retroalimentación), obliga a que no pueda usarse la misma clave, ya que sería fácil encontrar un patrón y descifrar el contenido del mensaje. Por este motivo se utiliza un generador de flujo de clave, de longitud igual o superior al mensaje.

Por tanto, los elementos de un cifrador de flujo son (Lucena López 2011):

1. Un *generador determinístico de clave (o generador de secuencia cifrante)*: a partir de una clave de inicialización secreta K produce una secuencia pseudoaleatoria de bits igual a la longitud del mensaje. Dicha secuencia de bits responde a una distribución uniforme, es empleada como clave tanto en el proceso de cifrado como en el de descifrado. Emisor y receptor deben contar con su propio generador de clave, los cuales producirán claves idénticas en ambos extremos de la comunicación.

2. Un *algoritmo de cifrado* basado en la función XOR. Este algoritmo servirá tanto para cifrar como para descifrar el criptograma, debido al carácter reversible de la función XOR.

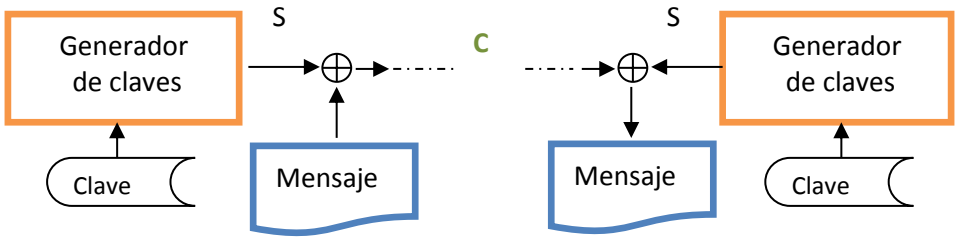
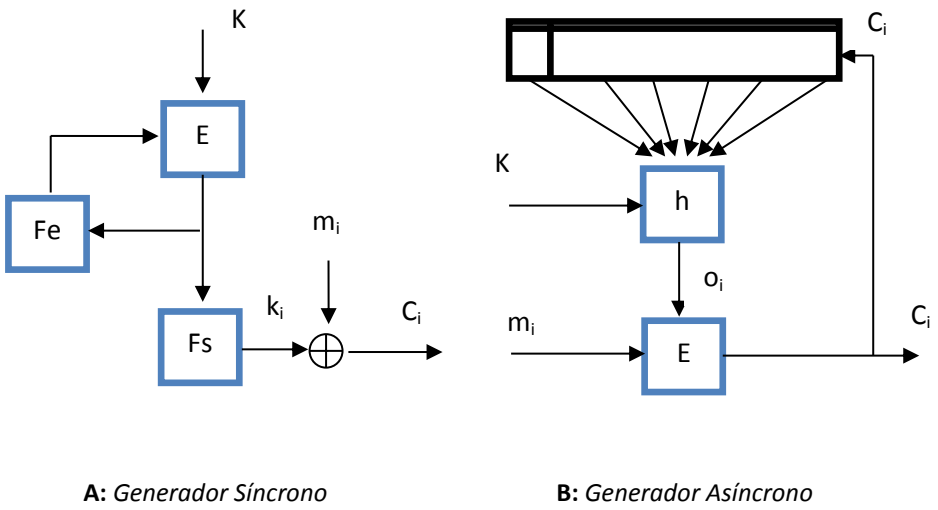


Figura 7. Esquema algoritmo simétrico de cifrado de flujo

Presenta resistencia a errores, al ser el cifrado independiente para cada elemento del texto en claro. Al no depender de otros elementos, destaca por su velocidad de cifrado. En cambio, es vulnerable dado que es posible alterar los elementos por separado y la difusión de los elementos en el criptograma es baja.

Los cifradores en flujo pueden ser clasificados de dos clases. La diferencia estriba en cómo se realiza la sincronía entre ambas partes, cifrador y descifrador, lo que conlleva diferencias en la estructura interna de ambas variantes:



A: *Generador Síncrono* **B:** *Generador Asíncrono*

Figura 8. Esquema de generadores de secuencia:
A: Generador Síncrono, B: generador asíncrono

Cifrado en flujo síncrono: la secuencia pseudo-aleatoria es independiente del mensaje. Para poder realizar el proceso de descifrado correctamente, tanto emisor como receptor deben compartir la señal de sincronización.

Si durante la transmisión se pierde o inserta algún bit, el resto del mensaje se descifrará de forma incorrecta a partir de ese fallo. En este caso, emisor y receptor deberán re-sincronizar sus generadores, evitando repetir alguna parte de la secuencia cifrada ya utilizada. Por otro lado, no se propagan errores de transmisión, si un bit resulta alterado durante la comunicación, solo descifrará incorrectamente esa posición afectada, sin afectar al resto del mensaje. Inserciones o borrados maliciosos son fácilmente detectables ya que interrumpen la sincronía. Desafortunadamente, no protegen frente a inversiones de bits al afectar únicamente al bit en cuestión y no provocar pérdida de sincronía. Esto obliga a emplear tanto técnicas de verificación que garanticen la integridad del mensaje recibido como de restablecimiento de la sincronía.

Cifrado en flujo auto-sincronizante: la secuencia pseudo-aleatoria se genera como función de la clave y de un número fijo de elementos del texto cifrado. En este caso no se necesitan señales de sincronización compartidas entre el emisor y receptor ya que en caso de pérdida de sincronía, ésta se recupera debido a la retroalimentación; el principal problema es la propagación de errores. Cualquier bit erróneo en la transmisión provocará que al descifrar se generen bits erróneos de secuencia cifrada, y por tanto, el mismo número de errores de descifrado del

texto en claro. Otra desventaja es que son altamente vulnerables a un ataque de inserción de mensajes, pero ello se puede evitar enviando mensajes adicionales de identificación de mensaje.

Una propiedad interesante de estos generadores es la dispersión de las propiedades estadísticas del texto claro a lo largo de todo el mensaje cifrado, ya que cada dígito del mensaje influye en todo el criptograma. Esto hace que los generadores síncronos se consideren en general más resistentes frente a ataques basados en la redundancia del texto en claro.

2.2.1.3. Resumen

En la siguiente tabla se puede observar un resumen de las principales ventajas y desventajas de las metodologías de cifrado en bloque y en flujo.

Tabla 1. Cuadro resumen algoritmos simétricos

	Ventajas	Desventajas
BLOQUE	Alta difusión de los elementos en el criptograma Es imposible introducir bloques extraños sin detectarlos Fácil implementación en software, al poder trabajar directamente con palabras del procesador	Baja velocidad de cifrado al tener que leer el bloque completo Propenso a errores de cifra; Un solo error se propaga por todo el bloque
FLUJO	Alta velocidad de cifra al necesitar tener en cuenta otros elementos Resistente a errores ya que cifra cada elemento se forma independiente Resultan ideales en sistemas hardware de tiempo real	Baja difusión de los elementos en el criptograma Es vulnerable tanto en cuanto los elementos pueden ser alterados por separado

2.2.2. Criptografía Asimétrica

Los algoritmos asimétricos utilizan dos claves complementarias llamadas clave privada y clave pública. Mientras que la clave privada debe ser conocida únicamente por su propietario, ya que es la base de seguridad del sistema, la clave pública es difundida abiertamente.

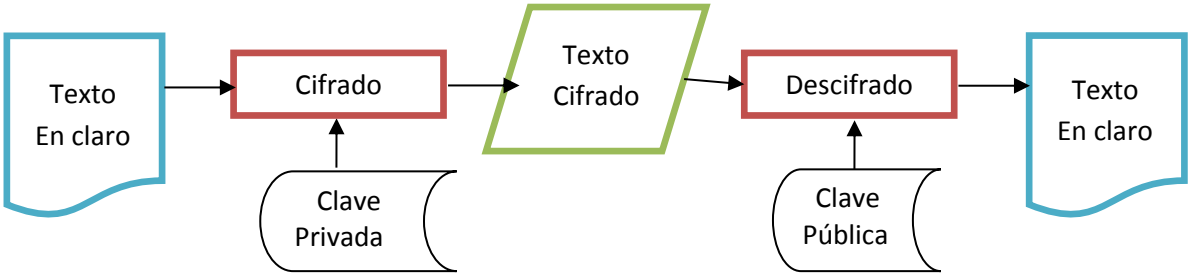


Figura 9. Esquema sistema criptográfico asimétrico

Para garantizar la seguridad de estos criptosistemas, se tienen que cumplir las siguientes condiciones (Álvarez n.d.):

- Conocido el texto cifrado no debe ser posible encontrar el texto en claro ni la clave privada
- Conocido el texto cifrado y el texto en claro debe resultar más costoso descifrar las claves que el posible valor de la información
- Conocida la clave pública y el texto en claro no se puede generar un criptograma correcto cifrado con la clave privada
- Dado un texto cifrado con una clave privada solo existe una pública capaz de descifrarlo, y viceversa
- Conocida una de las claves, debe resultar extremadamente difícil calcular la otra. O dicho de otra forma, cada pareja de claves debe ser única.

2.2.2.1. Clasificación de algoritmos asimétricos

La mayoría de estos sistemas apoyan su fortaleza en problemas matemáticos irresolubles, como lo son la factorización de enteros grandes o el problema del logaritmo discreto. Bajo esta premisa, existen principalmente tres familias de algoritmos asimétricos, atendiendo al principio matemático en el que basan su seguridad (Aguirre 2006):

1. *Algoritmos basados en la seguridad en el problema de la factorización entera.*

   Basan su seguridad en una debilidad de las máquinas de cómputo actuales: para un procesador es relativamente trivial el cálculo de enormes productos o potencias, pero el proceso contrario resulta muy costoso.

Así pues, la potencia de estos algoritmos reside en que no existe un método eficiente para factorizar números enteros muy grandes. A esta familia pertenecen RSA y RW (Rabin-Williams).

2. *Algoritmos basados en la seguridad en el problema del logaritmo discreto del grupo multiplicativo de un campo finito.*

   La potencia de los algoritmos basados en el problema del logaritmo discreto se basan en que no existe un método eficiente para calcular α a partir de la expresión

   $y = x^{\alpha} \pmod p$, donde p es un número primo.

   Matemáticamente se trata de un método bastante más complejo, pero computacionalmente es igual de complicado que el problema de factorización entera. A esta familia pertenecen DSA, DH, ElGamal y Nyberg-Rueppel.

3. *Algoritmos basados en la seguridad del logaritmo discreto sobre el grupo de puntos racionales de una curva elíptica sobre un campo finito.*

   Este grupo es un derivado del problema del logaritmo discreto, solo que en lugar de estudiarlo en un grupo multiplicativo, lo estudia en curvas elípticas.

   Estos algoritmos basan su potencia en que el cálculo de logaritmos sobre un sistema de curvas elípticas es computacionalmente más costos que su cálculo sobre cuerpos finitos. A esta familia pertenecen ECDSA y ECDH.

En comparación con los algoritmos simétricos, los algoritmos asimétricos presentan un mayor coste computacional que se observa en:

- El tamaño de las claves es mayor que las simétricas

  El tamaño de la clave es un aspecto muy importante ya que toda la seguridad recae en la complejidad de la misma, y en la imposibilidad de obtener la clave privada a partir de la pública, por lo que su tamaño puede variar entre los 512 bits y los 4096 bits.

- Para una misma longitud de clave y mensaje se necesita mayor tiempo de proceso

  La complejidad de cálculo que comportan los hace más lentos que los algoritmos de cifrado simétricos. Por ello, los métodos asimétricos se emplean para intercambiar la clave de sesión mientras que los simétricos para el intercambio de información dentro de una sesión.

- El mensaje cifrado ocupa más espacio que el original

  La mayor ventaja es que la distribución de claves es más fácil y segura. Con este sistema, se evita el problema de intercambio de claves existente en los sistemas

de cifrado simétrico. No es necesario que remitente y destinatario se pongan de acuerdo en la clave a emplear, simplemente intercambiar la clave pública, que puede ser utilizada por cualquiera que desee comunicarse con su propietario. Se necesitarán solo *n* pares de claves por cada *n* individuos que deseen comunicarse entre sí. No obstante, es necesario establecer un tercero en el proceso, autoridad de certificación, que confirme la titularidad de la clave pública de un usuario, es decir, que el único que posea la clave privada correspondiente sea el usuario auténtico al que pertenece.

2.2.2.2. Aplicaciones de los algoritmos asimétricos

Como ya se ha indicado, la información cifrada con una clave privada necesita su correspondiente clave pública para ser descifrada, y viceversa, lo cifrado con una clave pública sólo puede ser descifrado con su clave privada. Dependiendo del objetivo que se pretenda conseguir, se usará una clave u otra para codificar o decodificar (Lucena López 2011).

- *Protección de la información:* El emisor cifra con la clave pública del receptor el mensaje confidencial a transmitir. Cualquier intruso que intercepte la transmisión de un mensaje cifrado con la clave pública no podrá descifrar el contenido al no poseer la clave privada, pero sí podrá corromper la comunicación, conocido como ataque de intermediario, sustituyendo el mensaje dado que no se está identificando al origen.

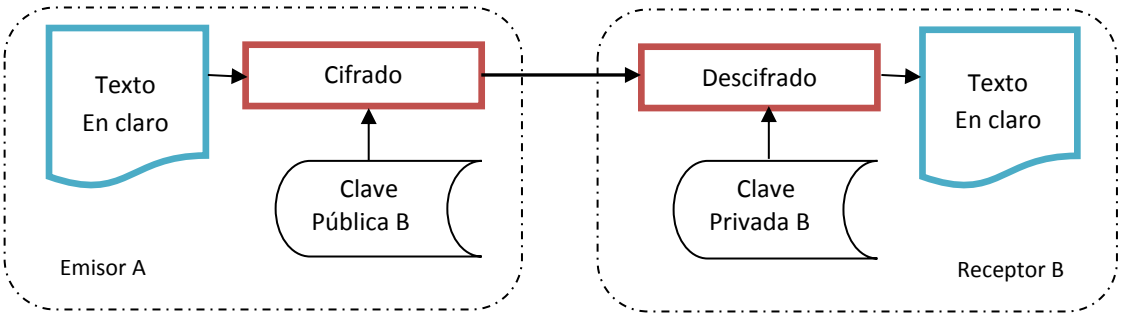


Figura 10. Transmisión de información

- *Autentificación de la información:* Si un usuario quisiera autentificarse a su destinatario, codifica un mensaje con su clave privada. El receptor puede decodificarlo ya que posee la clave pública del emisor, difundida abiertamente. Este modo de cifrado no proporciona confidencialidad, solo autentificación del origen, y define el fundamento de la *firma digital.*

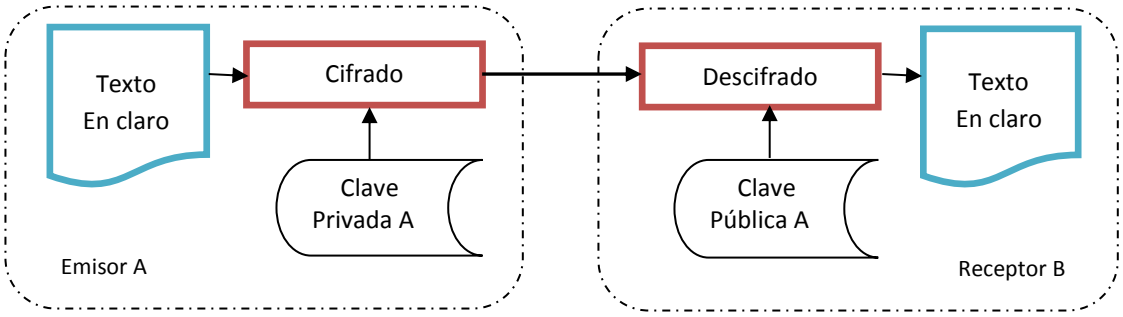


Figura 11. Autentificación de información

No todos los algoritmos asimétricos pueden proporcionar ambas funcionalidades, sino que tienen sus propias limitaciones. La siguiente tabla indica las aplicaciones que soportan los algoritmos asimétricos.

Tabla 2. Aplicaciones de algoritmos asimétricos

Algoritmo	Cifrado/Descifrado	Firma Digital	Intercambio de clave
RSA	Si	Si	Si
Diffie-Hellman	No	No	Si
ElGamal	Si	Si	No
DSA	No	Si	No
Curva Elíptica	Si	Si	Si

Como se ha visto hasta el momento, con los algoritmos asimétricos se consigue la autentificación de la información, es decir, poder identificar que un mensaje proviene de un emisor. Esta identificación debe hacerse empleando una función resumen, y no cifrando el mensaje completo.

2.2.3. Funciones hash o resumen

Consisten en la obtención de un resumen de longitud fija a partir de un mensaje de longitud variable. La función resumen es utilizada fundamentalmente para garantizar la integridad de la información, por lo que debe cumplir obligatoriamente estas propiedades:

- *Compresión*: independientemente del tamaño del mensaje origen, el resumen que se obtenga debe tener una longitud fija.
- *Facilidad de cálculo:* computacionalmente debe ser fácil calcular el resumen a partir del mensaje original.

Idealmente, el valor hash debería identificar de forma unívoca al mensaje. Esto no será siempre posible debido a que el conjunto de posibles mensajes a hashear es infinito, mientras que el conjunto de posibles valores es finito. Por tanto, es inevitable que la función devuelva en ocasiones el mismo hash para dos mensajes distintos. Este suceso se denomina colisión.

2.2.3.1. Clasificación de las funciones Hash

Atendiendo a una clasificación funcional, se diferencian dos tipos de funciones hash: Códigos de Detección de Modificaciones (MDC: *Modification Detection Codes*) y Códigos de Autentificación de Mensaje (MAC: *Message Autentification Codes*) (Menezes, Oorschot and Vanstrone 2011).

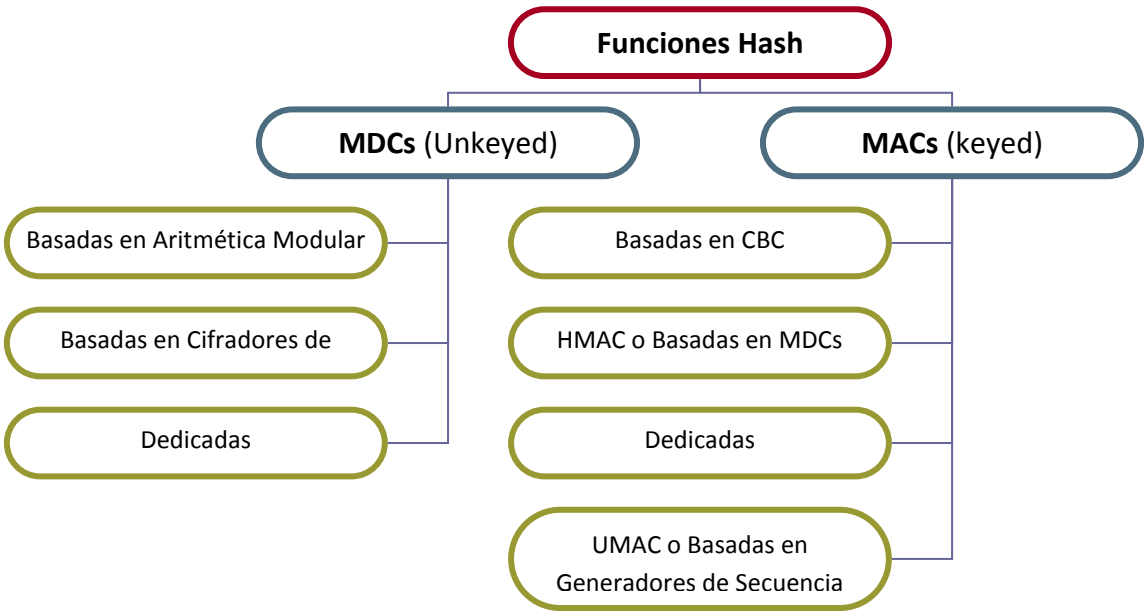


Figura 12. Taxonomía funciones resumen o hash

- MDC: realizan un resumen representativo de un mensaje (único parámetro de entrada), proporcionando, en conjunto con mecanismos adicionales, integridad de la información.

  Sea *x* un mensaje de entrada e *y* el resumen de ese mensaje, las funciones MDC pueden cumplir con estas propiedades adicionales a las básicas:

  - *Resistencia a preimagen* (o en terminología alternativa, "*one-way*"): conocido *y*, que sea computacionalmente irrealizable encontrar una entrada *x'* tal que y = h(x), siendo entrada *x* desconocida.

  - *Resistencia a segunda preimagen* (o en terminología alternativa, "*weak collision resistance*"): conocida la pareja *(x, h(x))*, que sea

computacionalmente irrealizable encontrar una segunda preimagen *x'* tal que *h(x')=h(x)*

- *Resistencia a colisión* (o en terminología alternativa, "*strong collision resistance*"): que sea computacionalmente irrealizable encontrar dos entradas cualesquiera *x,x'* tal que *h(x')=h(x)*

A partir del compromiso con estas propiedades, se diferencian dos tipos de funciones MDCs:

- OWHF: *One Way Hash Functions* (o en terminología alternativa, "*weak one-way hash functions*"): resistentes a encontrar una entrada que coincide con una salida determinada. Cumplen las propiedades de resistencia a preimagen y segunda preimagen.

- CRHF: *Collision Resistant Hash Functions* (o en terminología alternativa, "*strong one-way hash functions*"): resistentes a encontrar dos entradas que tengan la misma salida. Cumplen las propiedades de resistencia a segunda preimagen y a colisión.

Atendiendo a una clasificación estructural, existen diferentes tipos de MDCs:

- *Basadas en aritmética modular:* usan la aritmética modular como base de la función de compresión. Ejemplo MASH-1.

- *Basadas en cifradores de bloque:* reutilizan los cifradores de bloque de forma que se minimiza el esfuerzo de diseño e implementación y se aprovechan de la fiabilidad en su seguridad. Ejemplo Davies-Meyer, Matyas-Meyer-Oseas, Miyaguchi-Preenel.

- *Funciones hash dedicadas:* especialmente diseñadas desde el principio para este propósito. Ejemplo MD2, MD4, MD5, SHA-1, RIPEMD-160.

- MAC: Realizan un resumen representativo de un mensaje proporcionando integridad tanto en información como en el origen del mensaje, a partir de la aplicación de una clave secreta sin el uso de mecanismos adicionales.

  Además de cumplir con las propiedades obligatorias, garantizan una resistencia a computación que implica un no descubrimiento de la clave secreta. El ataque al que deben resistir es que sin conocer k, sea computacionalmente inalcanzable encontrar una nueva pareja x, $h_k(x)$ para un texto de entrada diferente $x \neq x'$ dado uno o más pares (xi, $h_k(xi)$). O dicho de otra forma, dado cero o más pares (xi, hk(xi)), es computacionalmente inalcanzable encontrar otro par (x, $h_k(x)$) para cualquier nueva entrada x=xi.

  Atendiendo a una clasificación estructural, se diferencian las siguientes MACs:

- *Basadas en modo CBC:* basados en cifradores por bloques. Cifran el mensaje empleando un algoritmo por bloques en modo de operación CBC. El valor del MAC será el resultado de cifrar el último bloque del mensaje. Ejemplo: CBC-MAC

- *HMAC:* basados en el uso de cualquier función MDC existente, aplicada sobre una versión del mensaje al que se ha añadido un conjunto de bits, calculados a partir de la clave que se quiere emplear.

- *Dedicadas:* diseñadas específicamente para este propósito. Ejemplo: MAA, MD5-MAC

- *UMAC:* basados en generadores de secuencia. El mensaje se parte en dos subcadenas, correspondientes al mensaje combinado con la secuencia y a la propia secuencia, cada una de las cuales alimenta un registro de desplazamiento retroalimentado. El valor del MAC se obtiene a partir de los estados finales de ambos registros. Ejemplo CRC-based MAC.

2.2.3.2. Aplicaciones de las funciones hash

Algunos de los usos de mayor relevancia de este tipo de funciones en criptografía son (Tovar 2010):

- Verificación de contraseñas: los datos de usuario y contraseña suelen ser codificados mediante funciones hash, de forma que si un usuario malicioso lograse acceder a esa información, no queda expuesta la información confidencial.

- Comprobación de integridad de mensajes y ficheros: aplicando la función hash sobre el mismo y entregando la huella resultante junto con el fichero original.

- Generación de números pseudoaleatorios: la idea es utilizar una cadena que ya exista para realizarle la función hash, y a partir de ese resultado generar el número pseudoaleatorio.

- Firma digital: las funciones hash son de gran utilidad a la hora de realizar una firma digital, ya que gracias a ellas los algoritmos de cifrado asimétricos no necesitan cifrar con su clave privada la totalidad del mensaje, si no solo el resumen del mismo, ahorrando grandes cantidades de cálculo.

2.2.4. Firma Digital

Una de las aplicaciones actuales más interesantes de la criptografía es la posibilidad real de añadir en un mensaje una firma digital alcanzando la autentificación completa. No

obstante, debido a la alta carga computacional de los sistemas de cifrado asimétricos, en vez de firmar el mensaje completo, el proceso de firma se realiza sobre un resumen o hash de dicho mensaje cifrado con la clave privada del emisor, el cual es mucho más ligero que el mensaje original.

Una firma digital es una secuencia de bits que se añade a una pieza de información cualquiera, y que permite garantizar la autenticidad de forma independiente del proceso de transmisión, tantas veces como se desee. Por tanto, ofrece el soporte necesario para la autentificación e integridad de los datos así como para el no repudio en origen, ya que relaciona de manera única al firmante con su firma.

La firma digital deberá cumplir las siguientes propiedades (Aguirre 2006):

- Será única, solo puede ser generada por su legítimo titular, y por tanto, infalsificable.
- Va ligada indisolublemente al mensaje. Una firma digital válida para un documento no puede ser válida para otro distinto.
- Es públicamente verificable. Cualquiera puede comprobar su autenticidad en cualquier momento, de forma sencilla.
- Será irrevocable, no rechazable por su propietario

La forma más extendida de calcular firmas digitales consiste en emplear una combinación de cifrado asimétrico y funciones resumen. El proceso de firma digital consta de dos partes fundamentalmente:

1. *Proceso de Firma*: el emisor A resume el mensaje en claro mediante una función hash y cifra el resultado con su clave privada, obteniendo su firma digital. Envía al destinatario B el mensaje original y su firma. Si se desea mantener la confidencialidad del mensaje, se cifra el mensaje original utilizando la clave pública de B.

2. *Proceso de Verificación de la Firma:* el receptor descifra asimétricamente el resumen mediante la clave pública de A, y aplica al mensaje original la misma función hash para obtener el resumen. Compara el resumen recibido con el obtenido, y si coinciden, se garantiza la autenticidad del emisor, la integridad del mensaje y el no repudio en origen.

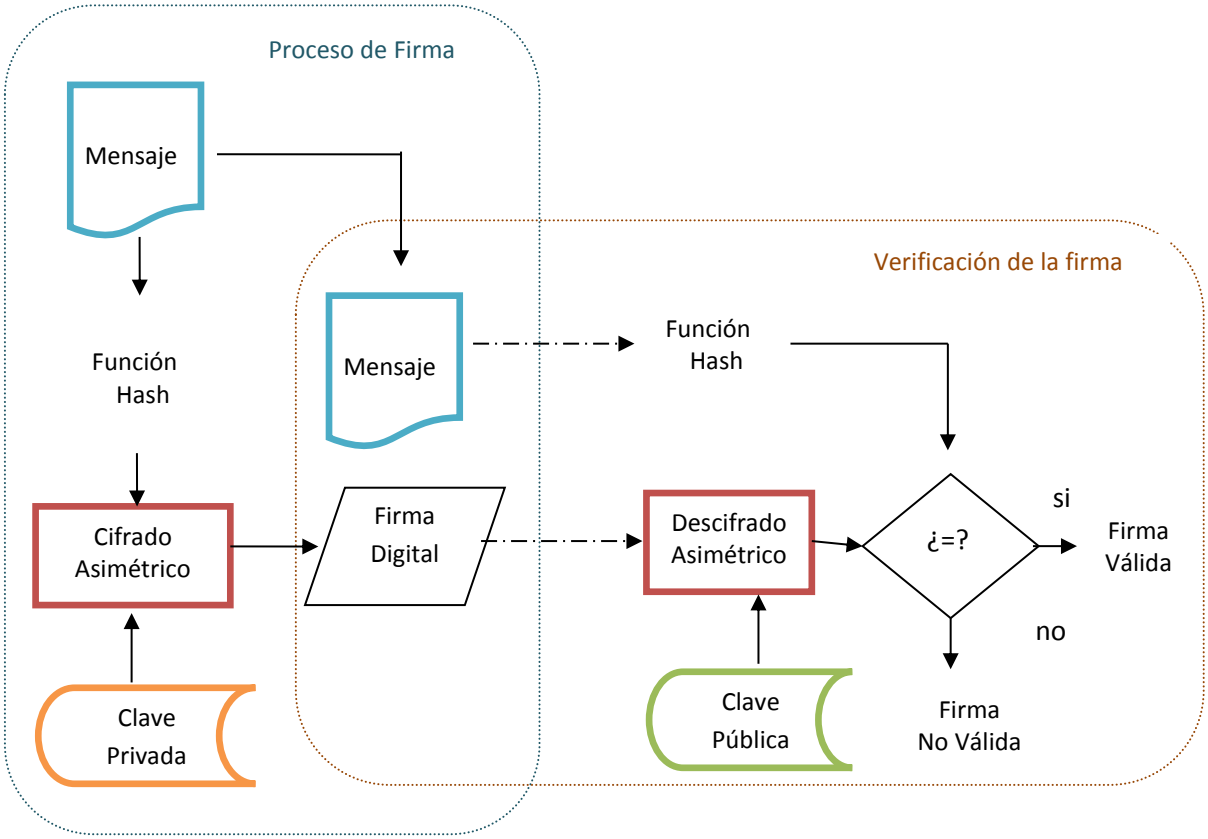


Figura 13. Esquema de la firma digital basada en funciones resumen y algoritmos de cifrado asimétrico

Resumen

Tabla 3. Cuadro resumen algoritmos simétricos vs. asimétricos

	Ventajas	Desventajas	Uso	Algoritmos más usados
Criptografía de Clave Secreta o Simétrica	Bajo coste computacional Alcanzan velocidades muy elevadas de cifrado en grandes volúmenes de datos Claves relativamente cortas y manejables Infraestructura sencilla Sistema eficiente en grupos muy reducidos, ya que solo es necesaria una única clave No es necesario disponer de una tercera parte confiable	Es necesario compartir la clave entre el emisor y receptor por canales que pueden no ser seguros Si se resuelve la clave, se compromete toda la comunicación No permite autentificar al emisor ya que una misma clave la utilizan varios individuos Se necesita un número elevado de claves: n*(n-1)/2, siendo n el número de individuos implicados en una comunicación cifrada Para maximizar la seguridad, se ha de cambiar la clave de forma frecuente, aumentando el problema de distribución de claves	Cifrado de grandes volúmenes de datos	DES, con tamaño de clave de 56 bits Triple DES, con tamaño de clave de 112 bits o 168 bits Blowfish, con tamaño de clave de 32 bits a 448 bits AES, con tamaño de clave de 128, 192 o 256 bits
Criptografía de Clave Pública o Asimétrica	Se transmite libremente la clave pública, mientras que la privada solo es conocida por el propietario Computacionalmente es complicado encontrar la clave privada a partir de la pública	Alto coste computacional, resultando muy lentos en el cifrado en volúmenes grandes de datos Gran tamaño del par de claves (orden de miles de bits) Necesidad de una gran infraestructura	Cifrado de mensajes cortos Firma digital Intercambio de claves	RSA, con tamaño de clave mayor a 512 bits (normalmente, 1024 bits) DSA, con tamaño de clave de 512 a 1024 bits ElGamal, con tamaño de

	Ventajas	Desventajas	Uso	Algoritmos más usados
	Permite autentificar a quien utilice la clave privada Número de claves reducido, ya que cada individuo necesitará únicamente un par de claves. Las claves se pueden mantener por más tiempo.	independientemente del número de individuos. Necesidad de una tercera parte confiable o autoridad de certificación en el proceso		clave comprendida entre los 1024 y 2045 bits

En la práctica, se utiliza una combinación de algoritmos entre la criptografía simétrica y asimétrica para poder dar seguridad a una gran parte de aplicaciones donde la transmisión de los datos se realiza por un canal considerado inseguro. Al realizar esta integración se compensan las desventajas de los tipos de cifrado y se utilizan las mejores características de cada uno, combinando la rapidez del cifrado simétrico con la facilidad de la administración de claves del cifrado asimétrico.

A continuación se va a realizar una descripción detallada de cada uno de los algoritmos más representativos de cada tipología.

2.3. Algoritmos de Cifrado Simétrico Por Bloques

2.3.1. DES: *Data Encryption Standard*

Es un algoritmo de cifrado simétrico de bloques de 64 bits y clave de longitud fija de 64 bits. (NIST 1999)

Consiste en la aplicación sucesiva de varias permutaciones y sustituciones (cambios de posición y de valor, respectivamente), entre el texto a cifrar y la clave, asegurándose al mismo tiempo que las operaciones puedan realizarse en ambas direcciones.

DES utiliza una clave de 64 bits, de los cuales 56 son usados para el cifrado, siendo los 8 restantes reservados para la detección de errores en el proceso. El punto débil de este algoritmo es que da lugar a un número limitado de claves posibles, que es 2^{56} (aproximadamente $7.2 * 10^{16}$), y por tanto, es susceptible de sufrir un ataque por fuerza bruta (probando todas las claves posibles).

Inicialmente el texto a cifrar se fracciona en bloques de 64 bits que se someten a una permutación fija inicial (IP). Posteriormente cada bloque se dividirá en dos bloques

iguales de 32 bits, Izquierdo y Derecho (denominados L y R), que serán sometidos a las siguientes acciones repetidas a lo largo de 16 iteraciones: $Li=R_{i-1}$; $Ri= L_{i-1} \oplus f (R_{i-1}, Ki)$

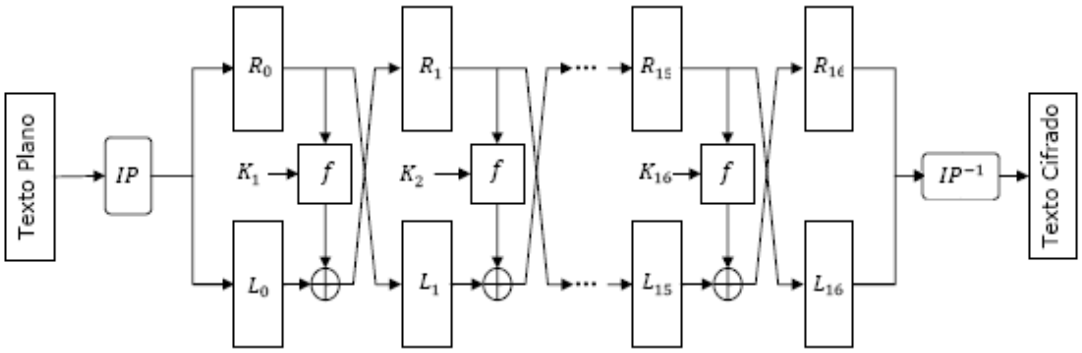


Figura 14. Esquema funcionamiento algoritmo DES

- Cada bloque Li y Ri se expande de 32 a 48 bits mediante la permutación de expansión
- Este bloque se combina con una subclave mediante una operación XOR
- El resultado es dividido en 8 bloques de 6 bits cada uno, para ser procesados por unas S_boxes o cajas de sustitución, cuya salida se codifica a 4 dígitos mediante una transformación no lineal. Estas cajas constituyen el núcleo de seguridad de DES.
- Las 32 salidas se reordenan de acuerdo a una permutación fija

Finalmente, se reconectarán las partes L y R, y se aplicará la inversa de la permutación inicial, (IP^{-1}).

La operación de descifrado se realiza de forma análoga a la del cifrado, excepto que las claves se usan en orden inverso (si el orden en cifrado fue k1, k2, k3… k16, en descifrado será k16, k15, k14… k1).

DES soporta los siguientes modos de operación: ECB, CBC, CFB y OFB, ya referenciados anteriormente, por lo que aunque en definición es algoritmo de bloques, si el cifrado se realiza bit a bit en modo OFB, puede actuar como cifrador de flujo.

Su principal ventaja es la rapidez de cálculo y la sencillez de su implementación.

Sus principales inconvenientes son la limitada longitud de clave que maneja, unido a la incapacidad de manejar claves de longitud variable y su debilidad en un uso continuado de la misma clave, puesto que disponiendo de un número suficiente de criptogramas, puede romperse la clave mediante criptoanálisis diferencial en 2^{47} iteraciones.

Desarrollado inicialmente por IBM en los años 70 bajo el nombre de Lucifer, fue adoptado por el NBS, ahora NIST y por la NSA en 1977 como estándar nacional, recogido en el FIPS 46-2, y en 1981 por la ANSI bajo el estándar X.3.92.

Aunque fue diseñado para resistir al criptoanálisis diferencial, un ataque por fuerza bruta en 1998 demostró que DES podría ser atacado en la práctica, y destacó la necesidad de un algoritmo de sustitución.

2.3.2. TipleDES

Surgido a raíz de cubrir la necesidad de seguridad de DES, no modifica el algoritmo, pero aumenta de forma efectiva el tamaño de la clave. Se encuentra recogido en los estándares FIPS 46-3 (NIST 1999) y ANSI x9.52.

Existen dos formas habituales de aplicar TripleDES:

1. Encadenando tres cifrados DES consecutivos mediante tres claves diferentes de 56 bits, equivalentes a usar una clave de 168 bits, también conocido como DES-EEE.

   Con tres claves K1, K2 y K3, aplicamos la transformación $C= Ek_3(Ek_2(Ek_1(m)))$,

2. Aplicando la siguiente transformación $C= Ek_3(Dk_2(Ek_1(m)))$, en la que Ek y Dk denotan cifrado y descifrado DES respectivamente. También es conocido como DES-EDE.

   Se aplica DES tres veces con tres claves distintas (k1, k2, k3). La primera vez en modo cifrado con la primera clave, la segunda en modo descifrado con la segunda clave, y la tercera en modo de cifrado con la tercera clave.

   A su vez, puede tener tres configuraciones en función de los valores de las claves:

   - Las tres claves son independientes
   - K1 y k2 son independientes, pero k1=k3. En este caso, la primera clave se usa la primera y última vez (k1=k3). Se codifica con k1, se decodifica con k2 y se vuelve a codificar con k1. La clave resultante es la concatenación de k1 y k2, con una longitud de 112 bits.
   - Las tres claves son la misma: K1=k2=k3.

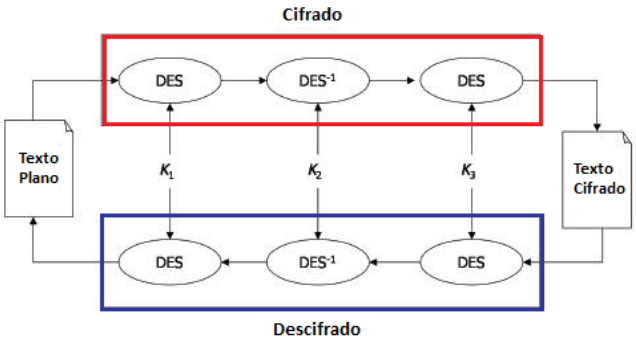


Figura 15. Esquema funcionamiento algoritmo TripleDES-EDE

Debido a la longitud de clave de 168, deja de ser vulnerable a un ataque por fuerza bruta y es altamente resistente al criptoanálisis al estar basado en DES. Sin embargo, el inconveniente principal es que este algoritmo es relativamente lento en su implementación software. El DES original se diseñó para implementaciones hardware de mediados de los 70 y no produce código software eficiente. El 3DES tiene tres veces más etapas que el DES, y por ello, es más lento.

2.3.3. IDEA: International Data Encryptation Algorithm

El algoritmo IDEA fue diseñado en 1992 para reemplazar el DES. Actualmente es de uso libre para fines no comerciales, aunque no está libre de patentes.

Es un algoritmo de cifrado simétrico de bloques de 64 bits que emplea una clave de longitud fija de 128 bits. Divide cada bloque en cuatro palabras de 16 bits que se combinan intercalando diferentes operaciones, como XOR, adicción y multiplicación modular, no existiendo operaciones a nivel de bit. Estas tres operaciones provocan confusión y no cumplen las leyes distributiva ni asociativa.

El proceso de cifrado consiste en ocho rondas completas de cifrado idéntico y una transformación final, también denominada media ronda. Se generan 52 subclaves de 16 bits a partir de la clave maestra de 128 bits. En cada una de las rondas se utilizan 6 subclaves diferentes, que se combinan con los bloques de cifrado mediante diferentes operaciones matemáticas ya mencionadas (XORs, sumas y multiplicaciones módulo 16). La estructura que crea la difusión es un bloque básico denominado Estructura MA Multiplication/Addiction. Usa solo dos claves por cada vuelta, y tanto sus entradas como sus salidas están conectadas por XOR.

Cuando ya se han realizado las ocho rondas idénticas, hay una transformación final que utiliza las últimas 4 subclaves para invertir la operación de cifrado inicial. Se denomina media ronda porque no atraviesa el bloque MA.

En la siguiente figura se pueden apreciar las operaciones que llevan a cabo en cada ronda:

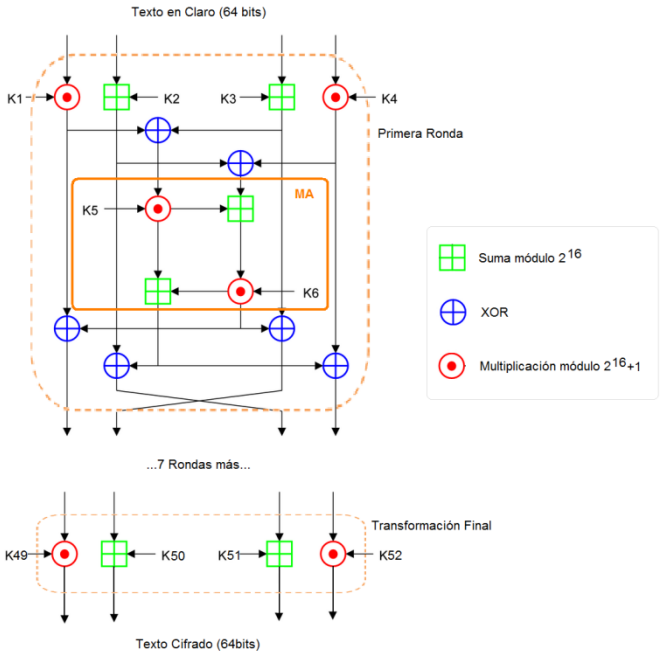


Figura 16. Esquema funcionamiento algoritmo IDEA

Debido a la longitud de clave hace imposible en la práctica un ataque por fuerza bruta. También se ha mostrado resistente a ataques de criptoanálisis diferencial bajo ciertos supuestos, por lo que constituye uno de los algoritmos simétricos de bloques más seguros disponibles en la actualidad. No obstante, se descubren una serie de claves débiles, que al ser fácilmente identificables, pueden eliminarse por diseño.

2.3.4. AES: *Advanced Encryptation Standard*

En respuesta a los ataques contra DES, la NIST lanzó un concurso para establecer el nuevo estándar avanzado de cifrado. El algoritmo Rijandael, nombre que se debe a sus dos autores belgas Joan Daemen y Vicent Rijmen, se anuncia como el sucesor del DES, publicándose el estándar final FIPS 197 en noviembre de 2001. (NIST 2001)

Rijndael sorprende por su rapidez, tanto en las implementaciones de software, como en las de hardware y además de ser relativamente fácil de implementar, requiere poca memoria para realizar los cálculos ya que está basado en una arquitectura algebraica con estructura de campo finito basada en redes de sustitución-permutación.

El procedimiento de codificación se basa en aplicar un número determinado de rondas a un valor intermedio denominado *Estado*, representado mediante una matriz de bytes rectangular, de 4 filas y Nb columnas (A[4, Nb]). La clave, con estructura análoga a la del Estado, está representada por una matriz de bytes de 4 filas y Nk columnas (K[4,

Nk]). El bloque que se pretende cifrar o descifrar se traslada byte a byte a la matriz de Estado y de forma homónima, la clave se copia sobre la matriz de clave.

Estrictamente hablando, AES es un caso particular de Rijndael. Rijndael maneja longitudes de clave y bloque variables, ambas comprendidas entre los 128 y los 256 bits. AES fija el tamaño del bloque a 128 bits, siendo flexible en la longitud de la clave, con valores permitidos de 128, 192 o 256 bits.

Dependiendo del tamaño de la clave que se emplee, AES realiza un número fijo de rondas tal y como muestra la tabla:

Tabla 4. Configuraciones posibles AES

	Clave 128 bits (Nk = 4)	Clave 192 bits (Nk = 6)	Clave 256 (Nk = 8)
Bloque de 128 bits Nb = 4	10 rondas	12 rondas	14 rondas

Estructuralmente se ha definido cada ronda como la composición de cuatro funciones invertibles diferentes, diseñadas para proporcionar resistencia frente a criptoanálisis lineal y diferencial. Cada una de las funciones tiene un propósito preciso:

- *Función ByteSub*: consiste en una sustitución byte a byte del bloque de entrada de acuerdo a unas tablas denominadas s-cajas con propiedades optimas de no linealidad.

- *Función ShiftRows*: consiste en realizar un desplazamiento de filas.

- *Función MixColumns*: consiste en realizar una mezcla de columnas.

  Ambas funciones realizan una mezcla lineal, que permite obtener un alto nivel de difusión a lo largo de varias rondas a través de las propiedades de permutación y sustitución.

- *Función AddRoundKey*: o función de adición de clave. Combina con un XOR el estado intermedio y la subclave correspondiente a cada ronda.

El esquema completo del proceso de cifrado con AES se puede observar en la figura 17:

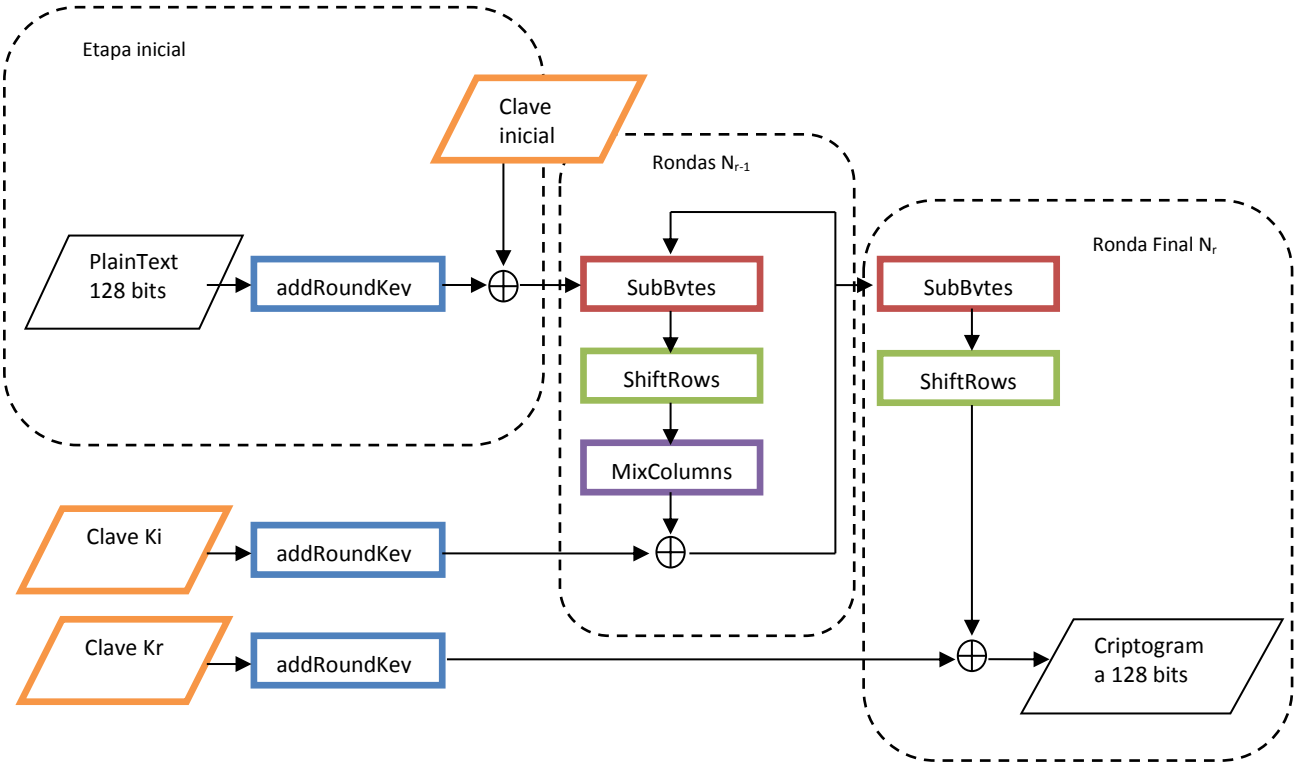


Figura 17. Esquema funcionamiento algoritmo AES

Puesto que cada ronda es una sucesión de funciones invertibles, el algoritmo de descifrado consistirá en aplicar las inversas de cada una de las funciones en el orden contrario, y utilizar los mismos Ki que en el cifrado, sólo que comenzando por el último.

Según sus autores es altamente improbable que existan claves débiles debido a la estructura de su diseño, que busca eliminar la simetría en las sub-claves. También se ha comprobado que es resistente a criptoanálisis lineal y diferencial. Por ejemplo, para realizar un ataque por fuerza bruta sería necesaria una búsqueda exhaustiva con 2^{120} operaciones, actualmente irrealizable.

2.3.5. BLOWFISH

Cifrador de bloques de 64 bits y clave variable, diseñado libre de patentes. (Blowfish n.d.)

- Longitud de clave variable: de 32 hasta 448 bits.

- Arquitectura de red tipo Feistel con 16 vueltas. En cada una de ellas se realiza una permutación función de la clave y una sustitución que es función de la clave y los datos, mediante operaciones como XOR, sumas modulares 32 y la utilización de cuatro S-cajas.

- Consiste principalmente en dos etapas: una inicial de expansión de la clave, donde se obtienen 18 sub-claves de 4.168 bits de longitud total, y otra de cifrado de datos.

- Características: compacto porque necesita sólo 5 KB de memoria, es muy rápido (5 veces más veloz que DES), es conceptualmente simple y su fortaleza puede variarse según longitud de la clave.

2.3.6. RC2

Cifrador de bloques de 64 bits y clave variable, diseñado por Ronald Rivest para la RSA Data Security Inc. Se encuentra recogido en el RFC 2268. (Rivest, A Description of the RC2(r) Encryption Algorithm 1998)

- Longitud de clave variable: de 8 a 128 bits.

- Arquitectura de red Feistel, que realiza 18 vueltas conocidas como mixing y mashing.

- Operaciones primitivas de cifra: suma modulo 32, operación OR exclusivo, complemento de bits, operación AND y rotación circular a la izquierda.

- Utiliza una tabla de sustitución denominada PITABLE en lugar de S-cajas y es casi tres veces más rápido que DES.

- Se usa en SMIME con longitudes de clave de 40, 64 y 128 bits.

2.3.7. RC5

Cifrador de bloques de tamaño variable, diseñado por Ronald Rivest para la RSA. (Rivest, The RC5 Encryption Algorithm 1997)

- Cifra bloques de texto de 32, 64 o 128 bits.

- Tamaño de clave hasta 2.048 bits, en función número de vueltas.

- Arquitectura de red Feistel, con un número de vueltas entre 0 a 255.

- Versiones específicas: RC5 –w/r/b donde w es el tamaño de la palabra (16, 32 ó 64 bits), r es el número de vueltas y b es el tamaño en octetos de la clave K. El valor estándar es RC5 –64/12/128.

- Rutinas de expansión de clave.
- Operaciones primitivas de cifrado basadas en suma módulo 2^w, OR exclusivo y rotación circular a la izquierda. Las rotaciones dependientes de los datos le fortalecen ante el criptoanálisis diferencial.
- Características: más rápido, arquitectura fácil de implementar, adaptable a procesadores de diferentes tamaños, requiere necesidades bajas de memoria y ofrece alta seguridad.

2.3.8. SAFER 64 Y 128

SAFER: Secure and Fast Encryption Routine, es un cifrador de bloques de 64 bits.

- Tamaño de clave: 64 o 128 bits.
- Número de vueltas de 0 a 10; mínimo recomendable 6.
- Operaciones de cifrado y descifrado distintas basadas en bytes, ya que cada bloque de texto a cifrar se divide en segmentos 8 bits.
- Cada vuelta tiene cuatro etapas: mezcla con la clave, mediante operaciones de XOR o suma modular 256; capa de sustitución mediante dos S-cajas; una segunda etapa de mezcla con las sub-claves; y una última capa de difusión que realiza operaciones lineales conocidas como Pseudo Transformaciones de Hadamard, PTH, cuyo objetivo es aumentar la difusión de los bits.
- Existen versiones SAFER SK-64 y SK-128 más seguras ante claves débiles que sus antecesoras (SK: Strengthened Key Schedule).

2.3.9. CAST 128

Cifrador de bloques de 64 bits con longitud de clave variable, libre de patentes para su uso, recogido en el RFC 2144 (Adams 1997)

- Tamaño de clave variable: de 40 hasta 128 bits en incrementos de octetos.
- Arquitectura de tipo red de Feistel con 12 o 16 vueltas.
- Usa ocho S- cajas basadas en funciones no lineales óptimas de tipo bent; cuatro de ellas se utilizan en procesos de cifra y las otras cuatro en la generación de claves.
- Operaciones básicas: suma y resta módulo 32, OR exclusivo y rotaciones circulares.
- Características: inmune a ataques por criptoanálisis diferencial y lineal; algoritmo estándar de cifra en últimas versiones de PGP.

2.3.10. SKIPJACK

Cifrador de bloques de 64 bits con una clave de 80 bits, desarrollado por la NSA, cuya implementación sólo está permitida en hardware. (NIST 1998)

- Arquitectura de red de Feistel no balanceada con 32 vueltas.
- Características: aunque los detalles del algoritmo no son públicos, se ha demostrado que es dos veces más rápido que DES. Su uso es adecuado en tarjetas inteligentes y su implementación es eficiente en hardware.

Cuadro resumen principales cifradores simétricos por bloques

Tabla 5. Resumen principales algoritmos simétricos por bloque

Algoritmo	Características	Ventajas	Desventajas
DES	Cifrado Bloques tamaño fijo 64 bits Clave longitud fija 64 bits (56 útiles + 8) Número de iteraciones: 16 Tipo Feistel	Rápido Fácil de implementar FIPS 46-2	Corta longitud de la clave Inseguro: se puede romper en 2^{47} iteraciones (criptoanálisis diferencial) o mediante ataques de fuerza bruta (2^{56} claves diferentes)
3DES	Dos claves de longitud fija 64-56 útiles- bits (una cifra y otra descifra) DSE-EEE2 y DSE-EDE2 Tres claves de longitud fija 64 bits, longitud efectiva 168 bits Dos configuraciones: DES-EEE3 o DES-EDE3 Tipo Feistel	Seguridad equivalente a un criptosistema de longitud de clave igual 112 btis Resistentes a ataques por fuerza bruta FIPS 46-3	Requiere más recursos para el cifrado y descifrado
IDEA	Cifrado en bloques de 64 bits Claves de longitud fija 128 bits Número de rondas: 8 Arquitectura isotopismo en cuasigrupos	El doble de rápido que DES Fácil de implementar Inmune ante criptoanálisis diferencial y ataques fuerza bruta	Patentado, aunque permite su uso para fines no comerciales Una clase de claves débiles
AES	Cifrado en bloques de 128 bits Clave de longitud variable	Rápido Fácil de implementar en HW	

Algoritmo	Características	Ventajas	Desventajas
	(128 / 192 / 256 bits)	y SW	
	Rondas flexibles (10/12/14) en función de la clave	Requiere poca memoria	
	Red algebraica GF (2^8): operaciones modulares a nivel de byte y de palabra de 4 bytes	Resiste a ataques de criptoanálisis diferenciales y lineales	

2.4. Algoritmos de Cifrado Simétricos de Flujo

2.4.1. RC4

Es un algoritmo de cifrado de flujo, diseñado por R.Rivest, para la RSA Security en 1987. Se difundió en internet una descripción que genera las mismas secuencias, por lo que también es conocido como ARCFOUR.

Su implementación es extremadamente sencilla y rápida. Emplea una caja de sustitución de 8x8 cuyas 256 entradas forman una permutación de los números del 0 al 255. La clave es de longitud variable y determina la permutación de la caja de sustitución.

Para su inicialización, requiere del funcionamiento de estos dos algoritmos:

1. KSA, *Key Scheduling Algorithm*: utiliza la clave secreta para inicializar la S-caja
2. PRGA, *Pseudo-Random Generation Algorithm*: genera una secuencia pseudoaleatoria

RC4 genera un flujo pseudoaleatorio de bits a partir de la clave original, y se combina con el mensaje en claro por medio de permutaciones y operaciones XOR. El descifrado es exactamente igual, debido a la simetría de la operación binaria.

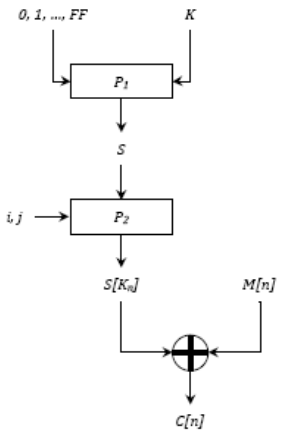


Figura 18. Esquema funcionamiento RC4

Aunque la RSA asegura su inmunidad a los criptoanálisis diferencial y lineal, fue excluido de los estándares de alta seguridad por presentar debilidades, como ser sensible a estudios analíticos del contenido de la S-caja; en una de cada 256 claves posibles, los bytes generados tienen una fuerte correlación con un subconjunto de los bytes de la clave. Una segunda vulnerabilidad permite la recuperación de la clave secreta si se cumplen unas premisas, y se intercepta el suficiente número de mensajes. (L, E and E n.d.)

2.4.2. A5

Es un algoritmo cifrador de flujo y en la actualidad existen tres versiones: A5/1, A5/2 y A5/3. A5/1 y A5/2 fueron definidos inicialmente por el estándar GSM. A5/3 está basado en el algoritmo Kasumi definido en 3GPP.

El cifrado está basado en la combinación de una serie de registros de desplazamiento realimentados linealmente (LFSRs) de diferente tamaño (tres en el caso de A5/1 con 19, 22 y 23 bits), con señales de reloj diferentes y una serie de combinaciones no lineales.

Aunque por definición la versión A5/2 es más débil que la A5/1, no se recomienda su uso debido a la facilidad de ataque al que queda expuesto el algoritmo, incluso en tiempo real. (Quirke 2004)

2.4.3. SEAL: Software Optimized Encryption Algorithm

Seal es un generador de secuencia diseñado en 1993 para IBM, por lo que está sujeto a patentes (U.S. Patent 5.454.039 y U.S. Patent 5.675.652).

A diferencia de algoritmos basados en sistemas lineales, presenta la posibilidad de acceso aleatorio, ya que define una familia de funciones pseudoaleatorias que calculan cualquier fragmento de la secuencia o únicamente a través de un número entero de 32 bits.

Emplea ocho registros de 32 bits y unos cuantos KBytes de memoria. Mediante un precálculo relativamente lento, genera un conjunto de tablas que permiten acelerar el algoritmo posteriormente. Para la generación de estas tablas se utilizan unas 200 llamadas a la función hash SHA, por lo que no es apropiado en situaciones en las que cambie la clave a menudo o no se disponga de espacio en memoria para las tablas.

2.5. Algoritmos de Cifrado Asimétricos

2.5.1. RSA

Es un sistema de cifrado de clave asimétrica válido tanto para cifrar como para firmar digitalmente. (RSA LAboratories 2002) Dado su comportamiento reversible, permite cifrar con la clave pública y descifrar con la privada si se desea confidencialidad, y viceversa, admite cifrar con la clave privada y descifrar con la pública si el objetivo es la autentificación.

Su seguridad radica en la dificultad para resolver el problema de factorización de grandes números enteros en la generación de las claves pública y privada.

Para un bloque de texto plano M y un bloque de texto cifrado C, el cifrado y el descifrado se definen a partir de la expresión:

$C = M^e \bmod n$

$M = C^d \bmod n = (M^e)^d \bmod n = M^{ed} \bmod n$

Tanto el emisor como el receptor deben reconocer los valores de n y e, y solo el receptor conocer d, de forma que la clave pública la forma el par KU={e,n} y la clave privada el par KR={d,n}. Para que este algoritmo sea satisfactorio para el cifrado, se deben cumplir los siguientes requisitos:

- Que sea posible encontrar valores de: e, d, n tal que

  $M^{ed} = M \bmod n$ para todo $M<n$

- Que sea relativamente fácil calcular M^e y C^d para todos los valores de $M<n$

- Que sea imposible determinar d dados e y n.

Los dos primeros requisitos se cumplen fácilmente. El tercero se puede cumplir para valores grandes de e y n.

La mayor dificultad reside en el *proceso* de generación de claves:

> Cada usuario elige aleatoriamente dos números aleatorios primos grandes distintos p y q, pero con longitudes en bits parecidas. La seguridad de este algoritmo reside principalmente en la elección de estos dos números; si p o q fueran compuestos, RSA no funciona.
>
> Se calcula el producto: n = pq.
>
> Se calcula la función de Euler (PHI) del módulo n: PHI(n) = (p-1)(q-1)

Se escoge un entero positivo e menor que PHI(n), que sea coprimo PHI(n): gcd(PHI(n),e)=1, 1<e<PHI(n)

Se determina un d que satisfaga la relación: d= e-1 (mod PHI(n))

La principal ventaja reside es que al no transmitirse la clave secreta, no es necesario tener claves diferentes para cada pareja de interlocutores. Como desventaja, no es tan eficiente a nivel de velocidad y resulta prácticamente inviable en transmisiones de cantidades de datos considerables. Puede presentar ciertos puntos débiles o vulnerabilidades:

- *Claves débiles:* existen casos para los cuales este algoritmo no altera el mensaje original, sea cual sea el valor de n si se cumple la condición $m^e = m \pmod n$
- *Claves demasiado cortas:* se recomienda el uso de claves superiores a 1024 bits. La elección de la longitud de la clave debe relacionarse con el tiempo que se desea que la información permanezca en secreto.
- *Ataques de intermediario:* se produce cuando un intruso expía una comunicación entre dos interlocutores, y se hace pasar por una de las partes. Para evitar esta situación, la clave pública se debe enviar firmada por una autoridad de certificación que confirme la autenticidad de la clave.
- *Ataques de texto en claro escogido:* Explota la posibilidad de que un usuario codifique y firme un único mensaje empleando el mismo par de claves.
- *Ataques de módulo común:* si empleamos p y q en todas las claves, un atacante podrá decodificar los mensajes sin necesidad de la llave privada.
- *Ataques de exponente bajo:* si e o d son demasiado bajos, un atacante puede romper el sistema. Se soluciona utilizando esquemas de relleno.
- *Firmar y codificar:* con RSA nunca se debe firmar un mensaje después de codificarlo, debe hacerse inicialmente.

2.5.2. ElGamal

Diseñado inicialmente para producir firmas digitales, su uso se extendió también para codificar mensajes. Basa su seguridad en el problema de los logaritmos discretos sobre el grupo multiplicativo de un cuerpo finito. No es un algoritmo determinístico, por lo que implica que para un mensaje dado se pueden obtener más de una firma válida.

Funcionamiento del algoritmo:

Generación de claves

Se escoge un número primo *n* y dos números aleatorios *p* y *x* menores que *n*. Se calcula la clave pública siguiendo la siguiente expresión:

$$y = p^x \pmod n$$

La clave privada corresponde al número x elegido.

Cifrado y Descifrado

Para *cifrar* el mensaje m se escoge un número primo aleatorio k menor que el módulo n y se calcula:

$$a = p^k \pmod n$$

$$b = y^k m \pmod n$$

El par (a, b) es el texto cifrado, de doble longitud que el texto original.

Para *descifrar* se calcula:

$$m = ba^{-x} \pmod n$$

Firma Digital y Verificación

Se escoge un número k aleatorio, tal que mcd(k,n-1) =1, y se calcula:

$$a = p^k \pmod n$$

$$b = (m-xa)k^{-1} \pmod{(n-1)}$$

La firma la constituye el par (a, b). En cuanto al valor k, debe mantenerse en secreto y ser diferente cada vez.

La firma se verifica comprobando que $y^a a^b = p^m \pmod n$

2.5.3. DSS: Digital Signature Standard

DSS es el estándar propuesto por el NIST para firmar digitalmente (FIPS 1994), dentro del cual se propone usar un algoritmo llamado DSA (Digital Signature Algorithm). Es decir, el algoritmo es parte del estándar.

El DSA es un variante de los algoritmos ElGamal y Schonoor, por lo que basa su seguridad en la resolución del problema matemático del logaritmo discreto y hace uso de la función hash SHA1. La longitud de la firma es de 320 bits para un resumen de 160 bits, con un tamaño de clave variable entre 512 y 1024 bits.

Funcionamiento del algoritmo:

Generación de claves

Por un lado se realiza la elección de los parámetros públicos (p, q, α, y), donde *p* es un número primo entre 512 y 1024 bits, *q* es un número primo de 160 bits divisor de (p-1); α es un generador de orden q del grupo p que verifica $\alpha = h^{\frac{p-1}{q}} \bmod p$, donde se cumple $1 < h < p-1$ y $h^{\frac{p-1}{q}} \bmod p > 1$;

La clave pública es el resultado de $y = \alpha^x \bmod p$

El parámetro *x*, número aleatorio entre $0 < x < q$, conforma la clave privada, por lo que su valor debe pertenecer en secreto.

Firma digital

La generación de la firma se calcula de acuerdo con las siguientes ecuaciones:

$r = (a^k \bmod p) \bmod p$;

$s = [k^{-1}(SHA(M) + xr)] \bmod q$;

siendo *k* un número aleatorio entre $0 < k < q$, y donde el par *(r,s)* conforma la firma digital.

Verificación

El destinatario efectuará una serie de operaciones, a partir de la clave pública (p, q, α, y) y la firma (r,s) que verifiquen la autenticidad de la firma.

2.5.4. DH: DIFFIE – HELLMAN

DH es el primer algoritmo de intercambio de claves seguro inventado en 1976, recogido en el RFC 2631 . Su finalidad es hacer posible que los usuarios intercambien de forma segura una clave secreta, a través de un canal de comunicación inseguro, para que pueda ser usada posteriormente en el cifrado de mensajes.

Su seguridad se basa en la dificultad de calcular el logaritmo discreto de un número en aritmética modular, comparado con la dificultad de calcular la exponenciación de un número en aritmética modular.

Funcionamiento del algoritmo:

Sean A y B los dos interlocutores de la comunicación:

- A y B seleccionan públicamente un grupo G de orden n, y eligen un número primo p y un elemento alfa $\alpha \in G$
- A calcula su clave pública a partir de la elección de un número aleatorio a, $\alpha^a \pmod p$, y se la envía a B
- B calcula su clave pública a partir de la elección de un número aleatorio b, $\alpha^b \pmod p$, y se la envía a A
- A y B calculan su clave privada compartida a partir de la expresión $K = (\alpha^a)^b \pmod p$

Como ventaja se observa que no son necesarias claves públicas en el sentido estricto, sólo una información compartida por los dos extremos de la comunicación.

Un intruso que descubra p y α e intercepte el valor $\alpha^a \pmod p$ que envía A o el valor $\alpha^b \pmod p$ que envía B, no podrá calcular los valores a y/o b ni el secreto.

No obstante, el algoritmo por sí solo no utiliza autentificación, por lo que su implementación está normalmente acompañada de la utilización de firmas digitales ya que de no hacerlo, se podría efectuar un ataque por fuerza bruta MITM.

Cuadro resumen algoritmos asimétricos

Tabla 6. Cuadro resumen principales algoritmos asimétricos

Algoritmo	Características	Ventajas	Desventajas
RSA	Seguridad basada en factorización grandes números enteros Número de iteraciones: 16 Tipo Feistel	Sirve tanto para cifrado como para la firma digital Muy rápido en la verificación de la firma	Poco eficiente en términos de velocidad de cifrado Es inviable su uso con grandes volúmenes de datos Gran tamaño de clave (entre 512 y 4096 bits) Necesario un mecanismo de certificación para asegurar la veracidad de la clave
ElGamal	Seguridad basada en problema de logaritmo discreto	Sirve tanto para cifrado como para la firma digital	Poco eficiente en términos de velocidad de cifrado Es inviable su uso con

Algoritmo	Características	Ventajas	Desventajas
			grandes volúmenes de datos
DSA	Seguridad basada en problema de logaritmo discreto Utilización de la función hash SHA-1	DSS es el estándar propuesto para firmar digitalmente	No codifica Muy lento en la verificación de la firma Gran tamaño de clave (entre 512 y 1024 bits)
DH	Algoritmo de Intercambio de claves seguro Seguridad basada en problema de logaritmo discreto	Solo es necesario compartir dos números sobre un canal inseguro	No firma ni cifra No autentifica

2.6. Funciones Resumen

Todas las funciones resumen o *hash* siguen el siguiente esquema básico de funcionamiento, descrito en la figura 19. Consiste en descomponer el mensaje en una serie de bloques de tamaño fijo y procesarlos uno a uno en varias etapas, generando salidas de alta entropía.

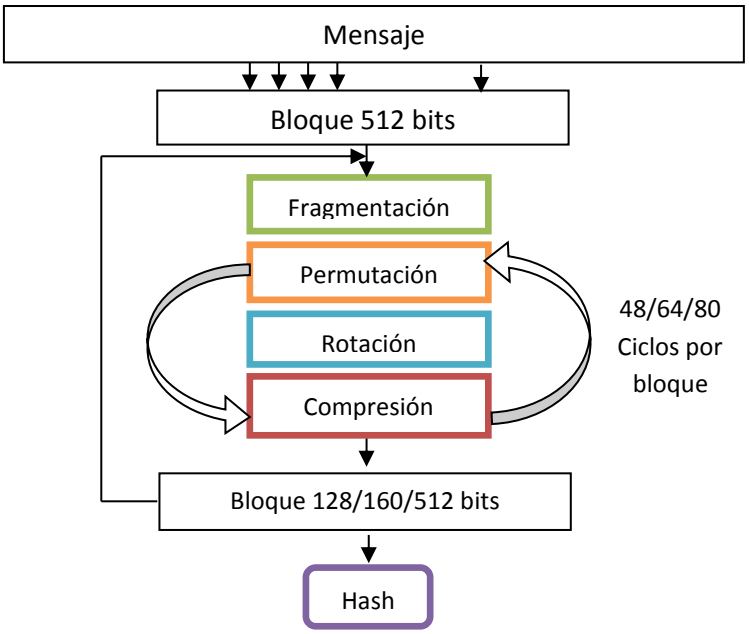


Figura 19. Esquema funcionamiento funciones resumen

2.6.1. MD4: Message Digest 4

Procesa bloques de entrada de 512 bits, para generar un resumen final de 128 bits, en 48 pasos.

Modo de funcionamiento:

- Pre-procesado: el mensaje se divide en bloques múltiplos de 512 bits, siguiendo un esquema de relleno, y se reservan los últimos 64 bits con el valor del tamaño del mensaje, empezando por el byte menos significativo.

- Procesado: Con los 128 bits de cuatro constantes ABCD de 32 bits cada una, y el primer bloque del mensaje de 512 bits se realizan diversas operaciones lógicas, definidas en tres funciones, entre ambos bloques. La salida de estas operaciones lógicas se convierte en el nuevo conjunto de cuatro vectores A'B'C'D', y realiza la misma función con el segundo bloque de 512 bits del mensaje, y así hasta el último bloque del mensaje.

  El bucle principal del algoritmo realiza 16 operaciones y se repite con cada bloque.

- Compresión: Al terminar, el algoritmo genera un resumen que corresponde a los últimos 128 bits de estas operaciones. El resultado es la concatenación de las constantes finales A, B, C y D.

2.6.2. MD5: Message Digest 5

Diseñado como una versión mejorada del MD4 para combatir las debilidades encontradas respecto a la resistencia a colisiones. (R.Rivest 1992)

Procesa los mensajes de entrada en bloques de 512 bits y produce una salida de 128 bits en 64 pasos, por tanto, su complejidad algorítmica es del orden de 2^{64}.

Modo de funcionamiento

- Preprocesado: igual que en MD4.
- Procesado: Esta sigue siendo la parte central del algoritmo, pero en este caso se definen cuatro funciones que se emplearan en las cuatro vueltas que se aplicarán sobre cada bloque.

  El bucle principal del algoritmo realiza también 16 operaciones y se repite con cada bloque. Dentro de cada vuelta, se realizan una serie de pasos coincidentes. La diferencia está en la función a utilizar.
- Compresión: Cada ronda combina el bloque de 512 bits del mensaje con el búfer estado, por lo que cada palabra del mensaje es usada cuatro veces. Después de las cuatro rondas de la función compresión, el búfer estado y el resultado se suman para obtener la salida. El valor hash de salida se obtiene de los registros A, B, C y D, donde el octeto más representativo es D y el que menos A.

Su seguridad se ha puesto en tela de juicio. Se ha desarrollado una serie de ataques criptoanalíticos que sugieren la vulnerabilidad del MD5.

2.6.3. SHA-1: *Secure Hash Algorithm*

Publicado en 1995 por el NIST como un estándar federal de procesamiento de la información (FIPS PUB 180-1)

Produce firmas de 160 bits a partir de bloques de 512 bits del mensaje original en un total de 80 vueltas, por tanto, su complejidad algorítmica será del orden 2^{80}, convirtiéndole en un algoritmo más seguro y resistente a ataques por fuerza bruta.

Modo de funcionamiento:

- Preprocesado: de la misma forma que los dos algoritmos anteriores.
- Procesado: utiliza cinco registros ABCDE de 32 bits cada uno, formando un bloque de 160 bits con el que se opera lógicamente con los bloques de 512 bits del mensaje. Cada bloque se divide en 16 palabras de 32 bits y se convierte en 80 trozos de 32 bits utilizando un algoritmo.

El bucle principal del algoritmo realiza 20 operaciones y se repite con cada bloque. Se definen tres funciones para las cuatro vueltas, y a cada ronda se utilizan cuatro constantes.

- Compresión: Al terminar, el algoritmo genera un resumen que corresponde a los últimos 160 bits de estas operaciones. El resultado es la concatenación de las constantes finales A, B, C, D y E.

Existen cuatro variantes más que se han publicado cuyas diferencias se basan en un diseño algo modificado y rangos de salida incrementados: SHA-224, SHA-256, SHA-384 y SHA-512.

2.6.4. RIPEMD 160

RACE Integrity Primitives Evaluation Message Digest

Se desarrolló en el proyecto RIPE (Europeo) de manos de un grupo de investigadores que lanzaron ataques con éxito parcial al MD4 y MD5.

Tiene una estructura muy similar a la de SHA-1. Produce un resumen de 160 bits a partir de bloques de 512 bits en 160 pasos (5 vueltas de 32 bits).

Es muy rápido, y su código fuente es abierto. Existe otra versión denominada RIPEMD-128.

2.6.5. Comparativa funciones Hash

En la siguiente tabla se muestra una comparativa con las cuatro funciones hash de mayor relevancia en la actualidad:

Tabla 7. Comparativa funciones hash

	MD4	MD5	SHA-1	RIMPD-160
Tamaño del resumen	128 bits	128 bits	160 bits	160 bits
Tamaño del bloque	512 bits	512 bits	512bits	512 bits
Número de pasos	48	64	80	160
Operaciones por vuelta	16/3	16/4	20/4	32/5
Número de vectores de inicialización	4 (32bits)	4 (32 bits)	5 (32 bits)	5 (32)
Fuerza de la preimagen	2^{128}	2^{128}	2^{160}	2^{160}
Fuerza contra ataque de colisión	2^{20}	2^{64}	2^{80}	2^{80}

2.7. Criptografía de Curvas Elípticas

La criptografía de curvas elípticas (ECC) es un enfoque a la criptografía de clave pública basado en la estructura algebraica de la curva elíptica sobre el campo finito. Su seguridad radica en la insolubilidad de los productos de multiplicación escalar del problema del logaritmo discreto elíptico, como se va a demostrar a continuación.

Las CE pueden definirse sobre cualquier cuerpo K. Si la característica de K no es ni 2 ni 3, entonces se define una curva elíptica sobre números reales como el conjunto de puntos que satisfacen la ecuación denominada ecuación reducida de Weierstrass:

E: $y^2 = x^3 + ax + b$, donde a,b son números reales/x,y $\in$ K.

En criptografía se requiere del uso de grupos algebraicos, conjunto de elementos con operaciones aritméticas definidas que pueden ser observadas geométricamente. Las CE pueden formar grupos algebraicos elípticos si se restringen los elementos del grupo, es decir, limitando el número de puntos de la curva, y la condición es que $4a^3 + 27\ b^2 \neq 0$. En esta casuística, la curva elíptica no tiene singularidades (vértice o intersección con ella misma). Si el discriminante se anulase sería una curva degenerada. Un grupo elíptico sobre números reales está constituido por todos los puntos que corresponden a la curva, junto con un punto especial O llamado punto en el infinito.

Los grupos elípticos son grupos aditivos, es decir, su función básica es la suma. Geométricamente se pueden observar las siguientes operaciones en estos grupos:

a. Suma elíptica: para calcular la suma de dos puntos, se traza la línea que atraviese P y Q, y se calcula el tercer punto de intersección de la recta con la curva. Se calcula el opuesto a este punto, obtenido tras trazar la recta paralela de este punto al eje de las coordenadas, obteniéndose la suma de los puntos P+Q, R.

b. El opuesto (o simétrico) de un punto Q(x,y) es su reflejo sobre el eje de las x: P=-Q=(x,-y). La recta vertical que les atraviesa no corta en un tercer punto. Por eso el grupo elíptico incluye el punto en el infinito O, que por definición se cumple P+O=P, y por extensión, P+(-P) =O.

c. Para calcular la suma de un punto sobre sí mismo, P+P=R, se traza la tangente del punto sobre la curva, y se calcula el opuesto del punto sobre el que ha cortado la recta.

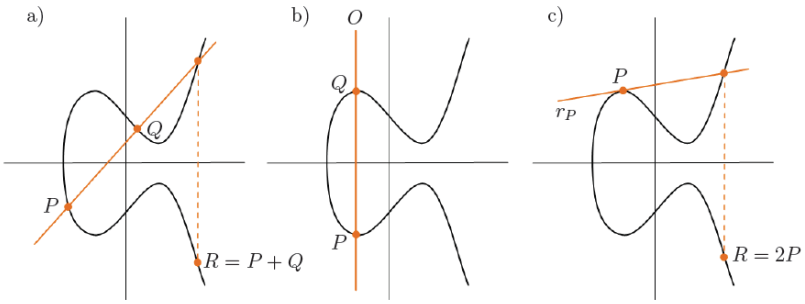


Figura 20. Operaciones con grupos algebraicos en CE

Aunque estas descripciones geométricas proporcionan una excelente forma de ilustrar las CE, computacionalmente se implementan con fórmulas algebraicas. Los cálculos sobre números reales son lentos e inexactos debido al error de redondeo. Dado que en las aplicaciones criptográficas se requiere rapidez y precisión algebraica, en la práctica se utilizan el grupo de curvas elípticas Fp, pertenecientes al campo finito primo y $F2^m$ pertenecientes al campo finito binario con 2^m elementos (Miret Biosca n.d.).

Grupos Elípticos sobre Fp

Fp es el campo finito contenido por *p* elementos primos, representados del 0 al (p-1), y que satisfacen la ecuación:

$y^2 \equiv x^3 + ax + b \pmod{p}$, donde $a,b \in$ Fp

Se forma un grupo elíptico si $x^3 + ax + b$ no contiene raíces dobles. A diferencia de grupos de CE sobre números reales, el número de puntos de un grupo elíptico sobre Fp es finito. El NIST recomienda cinco campos finitos con p del tamaño 192, 224, 256, 384 y 521. Como estas curvas consisten en un número determinado de puntos discretos, no es fácil observar la relación geométrica definida en los números reales, y por tanto, no se pueden aplicar las reglas geométricas. No obstante, es posible adaptar las propiedades aritméticas sobre Fp, de forma que ya no se produce error de redondeo.

En este caso, la seguridad de ECC depende de la capacidad de calcular en una curva elíptica E sobre un cuerpo finito Fp un generador P de un subgrupo cíclico G de puntos de E(Fp) y un punto Q de G, un entero n tal que Q = nP. (Dado el punto resultante Q, no es factible encontrar el múltiplo inicial P, la función es de un solo sentido. "n" es llamado el logaritmo discreto de Q en base P).

Grupos elípticos sobre $F2^m$

El campo finito $F2^m$ es la característica 2 del campo finito que contiene 2^m elementos. Aunque solo hay una característica 2 del campo finito $F2^m$ para cada potencia 2^m con $m \geq 1$, hay muchas formas de representar los elementos de $F2^m$, a continuación se representan como una combinación de polinomios binarios de grado m-1:

$\{a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1x + a_0 : a_i \in \{0,1\}\}$

Las reglas aritméticas pueden ser definidas por cualquier representación polinómica o con una representación óptima de base normal. La ecuación se ajusta para su representación binaria:

$y^2 + xy = x^3 + ax + b$, donde $a,b \in F2^m$

Un grupo elíptico sobre $F2^m$ incluye todos los puntos que satisfagan la ecuación, junto con el punto en el infinito O. En la práctica, el parámetro m debe ser lo suficientemente grande para impedir que el criptosistema se pueda romper. El NIST recomienda cinco campos binarios: 163, 233, 283, 409 y 571.

Los algoritmos criptográficos asimétricos DH y DSA se han adaptado con curvas elípticas, dando lugar a los algoritmos análogos ECDH y ECDSA respectivamente. La atracción principal de las CE en relación a RSA es que parecen ofrecer igual seguridad por un tamaño de clave mucho menor, reduciendo con ello los costes de procesamiento. Se ha demostrado (Certicom Research 2000) que para un tamaño de clave de 1024 en RSA, las CE solo necesitan 160 bits para alcanzar ese nivel de seguridad. Si el tamaño de la clave RSA se eleva a 2048 bits, el equivalente de CE es 224 bits.

La CCE es idónea para ser implementada donde el poder de cómputo y el espacio del circuito sea reducido, donde sea requiera una alta velocidad de procesamiento o grandes volúmenes de transacciones, y/o donde el espacio de almacenamiento, la memoria o el ancho de banda sea limitado.

En la actualidad existen varios estándares que incluyen el uso de la CCE, entre los cuales se encuentran: IEEE P1363, ANSI X9.62, ANSI X9.63, ISO 14888-3, ISO 15946, FIPS 186-2, IPSec (*Internet Protocol Security Protocol*)…

Por otra parte, aunque la teoría de la CCE está presente desde hace tiempo, el nivel de confianza todavía no alcanza al de RSA. Pero sin duda, el mayor inconveniente reside en su implementación, ya que Certicom posee más de 130 patentes, y es necesario poseer un amplio conocimiento en teoría de grupos y curvas elípticas para una implementación confiable y eficiente.

2.8. Protocolos de comunicación segura

Probablemente el objetivo principal que persigue la criptografía es el establecimiento de canales de comunicación seguros entre dos puntos, dado que la mayoría de los canales no son controlables por los interlocutores. En general, el mensaje es depositado en un medio hostil que se enfrenta principalmente a dos peligros:

- Acceso por agentes no autorizados: en un medio sobre el que no se puede ejercer ningún control, hay que garantizar que el mensaje resulte ininteligible a un atacante.

- Alteraciones en el mensaje: en un medio sobre el que no se puede ejercer ningún control, las alteraciones pueden aplicarse tanto sobre el mensaje propiamente dicho, como sobre la información acerca de su verdadera procedencia.

Como se ha visto hasta el momento, la criptografía proporciona mecanismos fiables para evitar ambos peligros. Cada aplicación de usuario exigirá unos requisitos determinados, por lo que la solución será realizar una combinación de algoritmos criptográficos que permita alcanzar el nivel de seguridad necesario. Esta combinación de algoritmos se estructura en forma de protocolos para proporcionar métodos de comunicación seguros y normalizados.

2.8.1. Protocolo TCP/IP

Internet está construido sobre los protocolos TCP (Protocolo de Control de Transmisión) e IP (Protocolo de Internet). El modelo de comunicaciones sobre el que está basado Internet se estructura en forma de capas apiladas, de manera que cada una se comunica con las capas inmediatamente superior e inferior, permitiendo intercambiar información de forma transparente (Lucena López 2011).

El modelo de referencia TCP/IP se organiza en 5 capas:

1. *Capa Física:* describe las características físicas de la comunicación.
2. *Capa de Enlace:* indica cómo viajan los paquetes de información a través del medio físico.
3. *Capa de Red:* se ubica el protocolo IP cuyo propósito consiste en hacer llegar los paquetes a su destino a través de una única red.
4. *Capa de Transporte:* se ubica el protocolo TCP que garantiza la llegada de los paquetes a destino, y en el orden correcto.
5. *Capa de Aplicación:* es la capa a la que acceden de forma directa la mayoría de las aplicaciones que utilizan Internet. Se reciben los datos de capas inferiores para ser enviados a destino. En este nivel pertenecen protocolos como HTTP, FTP, SSH, HTTPS, IMAP, DNS, SMTP, IRC…

En la práctica existen protocolos que proporcionan comunicaciones seguras en prácticamente todos los niveles de este esquema.

2.8.2. Protocolo TLS/SSL

Secure Sockety Layer (SSL) define el procedimiento para proporcionar comunicaciones seguras a través de Internet, de forma transparente al usuario y a las aplicaciones que lo

usan. Se sitúa en la capa de aplicación, directamente sobre TCP, y se usa principalmente para proporcionar seguridad a los protocolos HTTP (web), SMTP (email), FTP y NNTP (news).

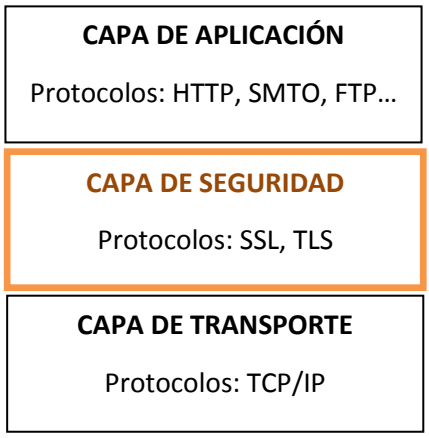


Figura 21. Pila de protocolos modelo OSI

Su fundamento consiste en interponer una fase de codificación de los mensajes antes de enviarlos por la red. Una vez establecida la comunicación, si una aplicación quiere enviar información a otra, la capa TLS/SSL la recoge, codifica, y la envía al destino a través de la red. Análogamente, el módulo TLS/SSL en destino se encarga de decodificar los mensajes y entregarlos como texto claro a la aplicación destinataria.

Está basado en la aplicación conjunta de criptografía simétrica, asimétrica, firma digital y certificados digitales para conseguir un canal o medio de comunicación seguro a través de la red:

- Con algoritmos simétricos realiza la codificación de los datos transferidos.
- Con algoritmos asimétricos realiza el intercambio seguro de las claves simétricas, resolviendo problema de confidencialidad en la transmisión de datos.
- La identidad del servidor web seguro (y/o del usuario cliente) se consigue mediante el certificado digital correspondiente, del que se comprueba su validez antes de iniciar el intercambio de datos sensibles, mientras que de la integridad de los datos intercambiados se encarga la firma digital en conjunto con funciones hash.

Una de las ventajas de emplear un protocolo de comunicaciones es que no está atado a ningún algoritmo, por lo éstos pueden ser reemplazados si en un futuro quedan comprometidos sin necesidad de modificar el protocolo. En la actualidad, la versión más actual de TLS/SSL usa los algoritmos simétricos de cifrado DES, TRIPLE DES, RC2, RC4 e IDEA, el asimétrico RSA y las funciones hash MD5 y SHA-1.

Otra ventaja se encuentra en la liberación de las aplicaciones para llevar a cabo las operaciones criptográficas antes de enviar la información, así como la transparencia que

permite su uso de manera inmediata sin modificar apenas los programas ya existentes. Por ejemplo, desde hace tiempo los principales navegadores incorporan un módulo TLS/SSL que se activa de forma automática cuando es necesario.

- *Arquitectura TLS/SSL*

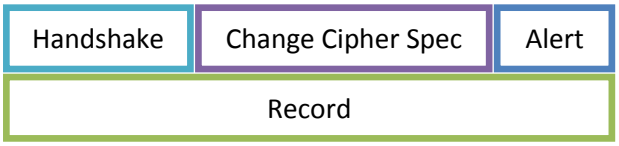


Figura 22. Arquitectura protocolos TLS/SSL

Está compuesto por dos capas:

- Una capa inferior en la que se encuentra el *Protocolo Record.* Encapsula todos los mensajes de alerta, cambio, especificación de cifrado, *handshaking* y de datos de usuario en un formato común y construye el canal de comunicaciones seguro, proporcionando una transferencia segura.

Una segunda capa superior formada por tres protocolos:

- *Protocolo Handshake*: se encarga de gestionar:

  - La negociación entre cliente y servidor de los algoritmos que se utilizarán en la comunicación.

  - El intercambio de claves y la autentificación basada en certificados digitales, utilizando una validación mediante una infraestructura de clave pública PKI (*Public Key Infrastructure*) cuando es necesario.

  - El cifrado del tráfico basado en criptografía simétrica. Genera una clave de sesión para la comunicación en función de los parámetros negociados.

  Una vez concluida la negociación de estos parámetros, comienza la conexión segura. Si alguna fase de la negociación falla, entonces la conexión no se establece.

- *Protocolo Change Cipher Spec:* consiste en un solo mensaje de 1byte que sirve para notificar cambios en la estrategia (o algoritmo) de cifrado.

  Por ejemplo, se puede usar en transacciones HTTP críticas en cualquier momento cuando uno de los dos interlocutores estima que la seguridad puede estar comprometida, y es posible volver a negociar la utilización de una nueva especificación de seguridad.

- *Protocolo Alert:* señaliza errores y advertencias en la sesión SSL establecida.

  Algunas de las circunstancias que se notifican son: mensajes no esperados, MAC incorrecto, error de descompresión o certificado corrupto o caducado.

- *Funcionamiento del protocolo*

Transport Layer Security (TLS) es un protocolo que proporciona autentificación, confidencialidad e integridad entre dos aplicaciones que se están comunicando. Basado en la versión 3.0 de SSL, incluye una serie de mejoras que se han recogido en el documento RFC 5246. No obstante, ya existen varias versiones de TLS (1.0, 1.1 y 1.2).

TLS intenta corregir las deficiencias observadas en SSL empleando una serie de medidas de seguridad adicionales como por ejemplo:

- La combinación de MD5 con SHA1 en la firma de documentación ha sido sustituida por una única función hash, de forma que se ha introducido un nuevo campo en el que se especifica el nombre del algoritmo usado.

- El control de números de versión del campo "encryptedpremastersecret" se hace más estricto.

- Si la versión del protocolo elegida por el servidor no es soportada por el cliente, el cliente envía una alerta de "protocol_version" y cierra la conexión.

- Las claves de sesión son calculadas de forma diferente.

- Introduce el uso de funciones HMAC, y se elimina el uso de los cifradores IDEA y DES.

- Mejora el sistema de alertas

- Establecimiento de una conexión segura

Una comunicación SSL/TLS consta fundamentalmente de dos fases:

1. Fase de saludo o handshaking. Consiste en una identificación mutua entre los interlocutores, para la cual se emplean habitualmente certificados X.509. Previamente, se negocian los algoritmos criptográficos que van a usarse en la comunicación, a partir del conjunto de algoritmos soportados por cada uno de los interlocutores. Tras el intercambio de claves públicas, los dos sistemas calculan la clave de sesión.

2. Fase de comunicación. En esta fase se produce el auténtico intercambio de información, codificado mediante la clave de sesión acordada en la fase de saludo.

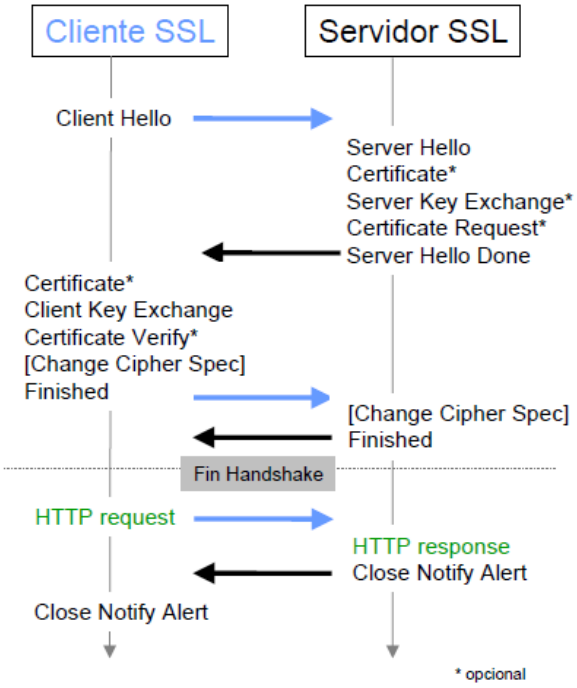


Figura 23. Establecimiento conexión TLS

Cada sesión lleva asociado un identificador único, que evita que un intruso escuche la comunicación y ataque a uno de los interlocutores (Ternero 2003).

Para evitar ataques de intermediario hay que garantizar que la clave pública pertenezca realmente a quien dice ser. La solución consiste en que esa clave esté firmada digitalmente por un tercero que certifique la autenticidad de la clave (Stallings 2003).

Lo ideal sería que cada usuario comunicara y comprobara de forma directa al resto de usuarios cuál es su clave pública, sin embargo, esto no es posible en la realidad por lo que se han desarrollado distintos esquemas para aportar confianza. Estos esquemas se pueden agrupar en dos tipos: *Esquemas Centralizados* y *Esquemas Descentralizados.*

> En los esquemas centralizados hay una arquitectura cliente-servidores, donde los servidores juegan un papel central y proveen servicios a los clientes. En general suelen ser más vulnerables a ataques de denegación de servicio.
>
> En los esquemas descentralizados hay varios nodos y cada uno tiene unas capacidades y derechos. Se consideran menos seguros contra ataques encaminados a publicar claves públicas falsas debido a que al haber varios nodos posibles a atacar, es más difícil asegurar su seguridad.

Los modelos más usados son:

- Uso de una infraestructura de clave pública o PKI.

  Hay una o varias entidades emisoras de certificados que aseguran la autenticidad de la clave pública y de ciertos atributos del usuario.

- Establecimiento de una web de confianza.

  Cada usuario es un nodo, que recoge las claves públicas de otros usuarios y asegura su autenticidad si están seguros de que la clave privada correspondiente pertenece en exclusiva a ese usuario. Un usuario puede directamente confiar en el conjunto de claves públicas en las que otro confía, ya sea directamente o a través de otras relaciones de confianza. En cada caso es el propio usuario el que decide el conjunto de claves públicas en las que confía y su grado de fiabilidad.

- Uso de criptografía basada en identidad.

  Se confía en una entidad denominada generador de claves privadas o PKG, que a partir de una cadena de caracteres identificativos del usuario genera un par de claves maestras privada y pública para ese usuario. Difunde la clave pública al resto de usuarios y la privada es comunicada en exclusiva al usuario a quien pertenece.

- Uso de criptografía basada en certificados.

  Es una mezcla entre la criptografía basada en identidad, eliminando algunas de sus deficiencias apoyándose en PKI. El usuario genera su propia clave privada y pública, pero la envía a una autoridad de certificación, que usando criptografía basada en identidad, genera un certificado que asegura la validez de los datos.

- Uso de criptografía sin certificados.

  Este modelo es similar al modelo que usa la criptografía basada en identidad, pero con la diferencia que lo que se genera en el centro generador de claves o KGC es una clave parcial. La clave privada completa se genera a partir de la clave privada parcial y un valor generado aleatoriamente por el usuario. La clave pública es generada también por el usuario a partir de parámetros públicos del KGC y el valor secreto escogido.

2.9. Infraestructura de Clave Pública

El modelo de confianza de una ICP (o PKI) sigue un esquema centralizado, basado en terceras partes confiables. Una PKI es una plataforma básica que engloba software, hardware, usuarios, políticas y procedimientos que posibilitan la gestión eficiente y confiable de las claves criptográficas y los certificados que pueden ser utilizados para propósitos de autentificación, integridad, confidencialidad y no repudio.

A su vez, una PKI se puede integrar o federar con otras, permitiendo comunicaciones seguras entre ellas. (MINETUR, Inteco, OSI n.d.)

Los elementos que componen una PKIs son:

- *Autoridad de Certificación*, CA, encargada de emitir y revocar certificados. Es la entidad de confianza que garantiza de forma unívoca y segura la identidad asociada a una clave pública.
- *Autoridad de Registro*, RA, gestiona el registro de usuarios y sus peticiones de certificación y/o revocación, así como los certificados respuesta a dichas peticiones, es decir, autoriza la asociación entre una clave pública y el titular de un certificado.
- *Repositorios*, estructuras encargadas de almacenar información. Los dos repositorios más importantes son el repositorio de certificados y el repositorio de listas de revocación de certificados, CRL: *Certificate Revocation List*.
- *Usuarios y entidades finales*, son aquellos individuos que poseen un par de claves y un certificado asociado a su clave pública.
- *Otras Terceras Partes Confiables*, como por ejemplo las Autoridades de Validación (VA) o de Sellado de Tiempo (TSA).

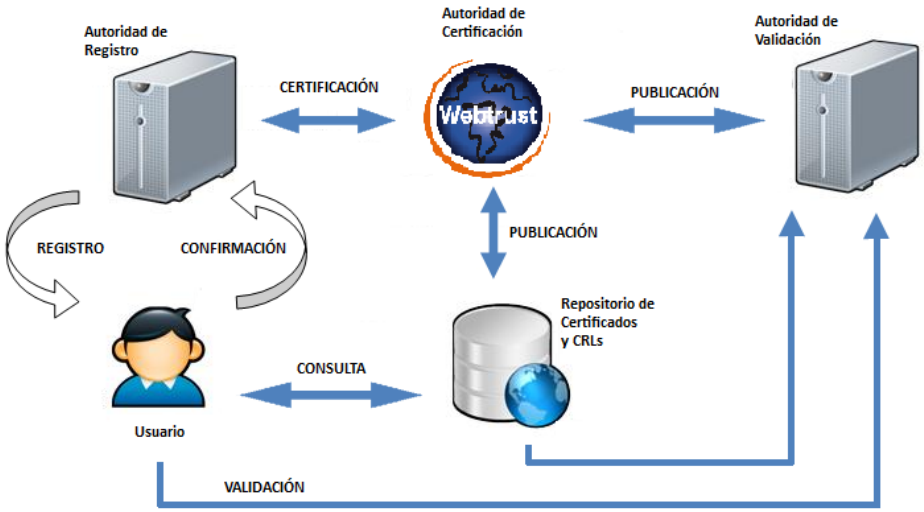


Figura 24. Elementos PKI

El mecanismo habitual de solicitud de un certificado digital consiste en que la entidad solicitante genera una petición CSR (Certificate Signing Request) que contiene la clave pública, junto con otros datos significativos, y la envía a la CA elegida. Ésta, tras verificar la información aportada, devuelve el certificado firmado al solicitante.

2.9.1. Certificado Digital X.509

El estándar X.509 solo define la sintaxis de los certificados, por lo que no está vinculado a ningún algoritmo en particular. (NIST 2008) A parte de la firma digital, contempla los siguientes datos:

- Versión del estándar, actualmente puede ser v1, v2 o v3.
- Número de serie: identificador del certificado, único para cada certificado expedido por una AC determinada
- Algoritmo empleado para la firma digital (RSA o DSA)
- Nombre del certificador, AC que ha firmado y emitido el certificado
- Periodo de validez
- Nombre del sujeto o propietario, identifica la identidad cuya clave pública está certificada en el campo siguiente. El nombre debe ser único para cada entidad certificada por una CA dada, aunque puede emitir más de un certificado con el mismo nombre si es para la misma entidad
- Clave pública del sujeto, representación de la clave pública en hexadecimal

Estos certificados se estructuran de forma jerárquica, de forma que el usuario puede verificar la autenticidad de un certificado comprobando la firma de la autoridad que lo emitió, que a su vez tendrá otro certificado expedido por otra autoridad de rango superior. De esta forma, se va ascendiendo en la jerarquía hasta llegar al nivel más alto, que deberá estar ocupado por un certificador que goce de la confianza de toda la comunidad.

Capítulo 3
Android

Android es una pila de software de código abierto, compuesta por un sistema operativo, establecido sobre el kernel de Linux 2.6, middleware y aplicaciones base, para su uso en dispositivos móviles.

Android es una plataforma móvil de código abierto (http://opensource.org/) distribuida bajo licencia Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0), y por tanto, de libre distribución. Permite al desarrollador el acceso completo al software del sistema, la modificación e incluso la distribución del mismo. De esta forma, es posible implementar aplicaciones que extiendan cualquier funcionalidad de los dispositivos. Fundamentalmente las aplicaciones están programadas en Java, aunque también se puede utilizar el lenguaje C/C++ para acceder a algunos componentes nativos de su arquitectura, como determinadas librerías o el núcleo.

Desarrollado inicialmente por la compañía Android Inc., que posteriormente fue comprada por Google, hoy forma parte de un estándar abierto desarrollado y promovido por la Open Handset Alliance (OHA), grupo de empresas de tecnología móvil, operadoras y distribuidoras, como HTC, Dell, Intel, Motorola, Qualcomm, Texas Instruments, Samsung, LG, T-Mobile o Nvidia, entre otras, aunque se mantiene el liderazgo de Google. Esta estrategia acelera la innovación en el sector de los dispositivos móviles, y permite que operadoras y fabricantes tengan mayor libertad a la hora de diseñar sus dispositivos.

3.1. Características principales

Entre sus principales *características*, cuenta con:

- Un framework de aplicación que habilita la reutilización y reemplazo de componentes
- Una máquina virtual Dalvik optimizada para móviles
- Un navegador integrado basado en WebKit
- Gráficos 2D optimizados por una librería gráfica propia y en 3D basados en la especificación OpenGL ES 1.0
- SQLite para almacenamiento de datos estructurados
- Soporte para gran variedad de archivos multimedia (MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF...)
- Soporte para un gran número de protocolos de comunicación: telefonía GSM, Bluetooth, EDGE, 3G y WiFi (4G, WiMAX...)
- Proporciona un entorno de desarrollo completo (SDK), incluyendo emulador de dispositivos (AVD), herramientas para la depuración, análisis de memoria y rendimiento (DDMS) y plugin para el IDE Eclipse

Si bien Android ofrece una cantidad de herramientas muy útiles como una API potente, versatilidad de aplicaciones, multitarea y posible sustitución de funcionalidades originales del sistema operativo por otras personalizadas, cuenta con alguna desventaja inherente a su categoría de software libre:

- *Fragmentación*: la gran variedad de dispositivos disponibles con distintas capacidades y distintas versiones de Android puede complicar el desarrollo de las aplicaciones, si bien es cierto que hay formas de especificar las necesidades mínimas de una aplicación.
- *Obsolencia dependiente*: cada fabricante puede decidir no seguir dando soporte a modelos antiguos aunque estos tengan capacidad para hacer funcionar las últimas versiones.
- *Personalización*: las operadoras suelen modificar las aplicaciones instaladas en los dispositivos, moldeándolos a sus necesidades en lugar de orientarlas al usuario, llegando a extremos de imposibilitar a éste la desinstalación de dichas aplicaciones independientemente de las necesidades del mismo.

A pesar de los posibles inconvenientes de un proyecto de software libre, la comunidad de desarrollo de Android crece a pasos agigantados, permitiendo, por ejemplo, disponer de funcionalidades de ultima generación en teléfonos antiguos. Es más, la libertad de uso y que, sin pertenecer a la OHA, muchos fabricantes puedan utilizar Android, ha logrado que existan dispositivos en el mercado que se adapten a las necesidades de cada consumidor.

3.2. Evolución de las versiones

Desde que en agosto de 2008 apareciera la versión 0.9 del SDK de Android hasta su versión actual, han salido diferentes versiones que han ayudado a la implementación de nuevas funciones. En la siguiente tabla se visualizan las plataformas lanzadas hasta la fecha, que se identifican de tres formas alternativas: versión, nombre comercial y nivel de API: (Android n.d.)

Tabla 8. Versiones de plataformas. Actualización a 4 de septiembre de 2013

Versión		Nombre	API Level	Distribución
1.5		Cupcake	API Level 3, NDK1	nd
1.6		Donut	API Level 4, NDK 2	nd
2.1		Eclair	API Level 7, NKD 3	nd

Versión		Nombre	API Level	Distribución
2.2.x		Froyo	API Level 8, NDK 4	2.4%
2.3 - 2.3.7		Gingerbread	API Level 9, 10 10zNDK 5	30.7%
3.1 - 3.2.x		HoneyComb	API Level 12, 13 NDK 6	0.1%
4.0.3 - 4.0.4		Ice Cream Sandwich	API Level 15, NDK 8	21.7%
4.1.x - 4.2.x		Jelly Bean	API Level 16, 17	45.1%

Siempre que se ha lanzado una nueva plataforma, se ha diseñado de forma compatible con las versiones anteriores. Es decir, solo se añaden nuevas funcionalidades, y en caso de ser necesaria la modificación de alguna, no se elimina sino que se etiquetan como obsoletas, aunque se permite su uso.

3.3. Arquitectura

La Figura 25 resume los componentes de seguridad y las consideraciones de los diferentes niveles de la pila de software de Android. Cada componente asume que los componentes inferiores están adecuadamente asegurados, con la excepción de un pequeño código de *root*, todo el código por encima del Kernel de Linux (nivel inferior) está limitado por el *Sandbox* (Android Developers n.d.).

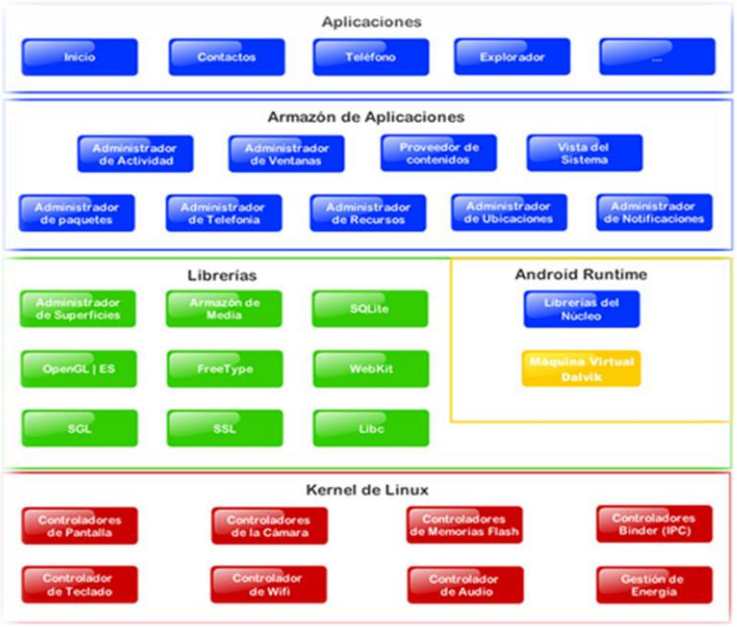


Figura 25. Arquitectura de Android

- Android Linux Kernel

  Es el nivel inferior, basado en la versión 2.6 del núcleo de Linux, que sirve como capa de abstracción entre el hardware y el resto de la capas de la arquitectura de Android. Se encarga de gestionar los servicios de seguridad, gestión de memoria, administración de procesos, protocolo de red o drivers del sistema.

  Solo por adoptar el Kernel de Linux, Android cuenta con las siguientes características inherentes al nivel de seguridad:

  - Modelo de permisos basado en usuarios
  - Aislamiento de procesos
  - Mecanismos extensibles para asegurar IPC (Inter-Process Communications)
  - Posibilidad de eliminar partes innecesarias y potencialmente inseguras del Kernel

  Por tanto, proporciona seguridad tanto a nivel de sistema operativo como a nivel de comunicaciones entre procesos. De esta forma, se permiten comunicaciones seguras entre aplicaciones que corran en diferentes máquinas virtuales. Si detecta que la seguridad del sistema puede quedar comprometida, aísla la aplicación para evitar daños en otras aplicaciones, en Android o en el propio dispositivo.

- Entorno de ejecución Android (Android Runtime)

  Compuesto por una pequeña librería propia del Kernel junto con la máquina virtual Dalvik. La máquina virtual Dalvik, basada en registros, depende del Kernel para ciertas funcionalidades como la gestión de los hilos o de la memoria.

  Cada aplicación Android corre su propio proceso, con su propia instancia de la máquina virtual Dalvik. Ésta ha sido diseñada para poder ejecutar múltiples máquinas virtuales de manera eficiente y optimizada para entornos con poca memoria y poca potencia.

- Librerías Android

  Incluye un conjunto de librerías en C/C++ que proporcionan la mayor parte de las capacidades de Android. Entre las librerías más importantes, se pueden destacar:

  - *Librería de System C:* Una implementación tipo BSD de la biblioteca estándar de C (libc), adaptada para su uso en sistemas Linux embebidos.
  - *LibWebCore:* Un motor moderno que es parte tanto del navegador principal

como de los embebidos.

- *Librerías multimedia:* Basadas en OpenCore, dan soporte a la reproducción y grabación de audio y vídeo, y formatos de imágenes estáticas.
- *Librerías Gráficas:* Implementación basada en las APIs de OpenGL/SL y SGL, y por tanto, sustentan la capacidad gráfica de Android con el manejo de gráficos 3D y 2D respectivamente.
- *Librería SQLite:* Proporciona un motor de bases de datos relacional ligero y potente disponible para todas las aplicaciones
- *Librería SSL:* Posibilita la utilización del protocolo SSL para establecer comunicaciones seguras.

- Framework de aplicación (marco de trabajo)

  Compuesto por un conjunto de herramientas para el desarrollo de aplicaciones, tales como el acceso a la misma API utilizada en las aplicaciones nativas. El diseño de esta arquitectura facilita la reutilización de componentes; cualquier aplicación puede publicar sus capacidades y otra aplicación podrá hacer uso de las mismas (sujeto a reglas de seguridad del *framework*). Este mismo mecanismo permite que los componentes sean reemplazados por el usuario.

  En este nivel, encontramos algunos de los servicios disponibles:

  - *Activity Manager,* gestiona el ciclo de vida de las aplicaciones y establece un sistema de navegación entre las mismas.
  - *Views System*, proporciona un gran número de elementos para construir interfaces de usuario, como listas, cajas de texto, botones o incluso un navegador embebido.
  - *Content Providers*, permiten a las aplicaciones acceder a datos de otras aplicaciones o compartir los suyos propios.
  - *Resource Manager*, da acceso a recursos como imágenes, cadenas de texto o los archivos XML en los que se especifica el diseño de la interfaz.
  - *Notification Manager,* permite a todas las aplicaciones mostrar alertas en la barra de estado.

- Aplicaciones

  Android se distribuye libremente con un conjunto de aplicaciones básicas incluyendo cliente de email, programa de SMS, calendario, mapas, navegador y agenda de contactos entre otros. No obstante, en este nivel también estarían

incluidas aquellas que el usuario vaya añadiendo posteriormente, ya sean propias o desarrolladas por terceros.

Componentes Básicos de Alto Nivel de las Aplicaciones

Son los bloques esenciales para la construcción de una aplicación. Existen cuatro tipos de componentes, cada uno con un propósito y ciclo de vida distinto:

- Activity: Representa la interfaz de usuario de una única pantalla.
- Service: Representa una tarea sin interfaz gráfico que se ejecuta en segundo plano por un periodo indefinido de tiempo.
- ContentProvider: Se encarga de manejar datos compartidos entre aplicaciones, de un fichero, de una base de datos SQLite o en cualquier otro formato, que pueden ser consultados o modificados.
- BroadcastReceiver: Son componentes que responden a un determinado evento, originado por el sistema u otras aplicaciones. No tiene interfaz gráfico pero puede enviar notificaciones, como por ejemplo, a la barra de estado del sistema.

Antes de poder usar estos componentes es necesario que estén declarados en el archivo *AndroidManifest*, en el cual también deben ir incluidos los permisos que necesita la aplicación, el nivel de API mínimo que se requiere, requerimientos hardware y software necesarios. Esta información es esencial, y se debe conocer antes de poder ejecutar la aplicación.

Por defecto, una aplicación solo tendrá acceso a un número limitado de recursos. El sistema gestiona el acceso de recursos para que en caso de un uso incorrecto o malicioso, no pueda impactar negativamente en la experiencia del usuario, la red o los datos del dispositivo.

Estas restricciones están implementadas de diferentes formas. Algunas capacidades están restringidas de forma intencionada con carencia de API, como por ejemplo en la manipulación de la SIM. En otras ocasiones, el API sensible de uso está limitado a aplicaciones confiables y protegidas a través de un mecanismo conocido como permisos. Entre estas APIs nos encontramos la funcionalidad de la cámara, localización por GPS, Bluetooth, SMS/MMS funciones y conexiones de red.

3.4. Modelo de seguridad

Android procura ser el sistema operativo de dispositivos móviles más seguro y usable

reutilizando controles de seguridad de sistemas operativos tradicionales para:

- Proteger los datos del usuario
- Proteger los recursos del sistema (inclusive la red)
- Aislar las aplicaciones.

Para lograr estos objetivos, Android provee las siguientes medidas de seguridad, la mayoría derivadas de los estándares del Kernel Linux:

- Seguridad robusta en el nivel del SO a través del Kernel de Linux
- *Sandbox* obligatoria para todas las aplicaciones
- Permisos definidos de aplicación y de usuario
- Comunicación entre procesos segura
- Firma de aplicaciones

Como ya se ha mencionado anteriormente, Android es un sistema multiproceso, en el que cada aplicación se ejecuta en su propio proceso (o máquina virtual). La seguridad entre las aplicaciones y el sistema se aplica, generalmente, en el nivel de proceso a través de estándares definidos por Linux, tales como la identificación de usuarios y grupos que se asignan a las aplicaciones (1). No obstante, se definen características adicionales de seguridad, por un lado, a través de un mecanismo de permisos que impone restricciones a ciertas operaciones que un determinado proceso puede realizar (2), y por otro, concediendo permisos de acceso ad-hoc a ciertos datos mediante un mecanismo de URIs (*Uniform Resource Identifiers*) (3) (Android n.d.).

(1) Identificación usuarios y grupos

Android emplea las ventajas de Linux a nivel de proceso en el sentido de identificación y aislamiento de los recursos de las aplicaciones. Asigna una identificación única (UID) a cada aplicación, y ésta se ejecuta en un proceso (o entorno seguro de ejecución) separado del resto, lo que se denomina el Sandbox de aplicaciones. Por defecto, las aplicaciones no pueden interactuar con el resto y tienen acceso limitado al sistema operativo.

Como el Sandbox se encuentra en el Kernel, este modelo de seguridad se extiende desde el código nativo a las aplicaciones del sistema operativo. A pesar de todo, el Sandbox no es inmune a la corrupción. No obstante, para romperlo en un dispositivo correctamente configurado, se debe comprometer la seguridad del Kernel de Linux.

(2) Permisos entre procesos

Un punto clave del diseño de la arquitectura de seguridad de Android es que, por defecto, ninguna aplicación tiene permisos para realizar operaciones que tengan un impacto negativo en otras aplicaciones, en el sistema operativo, o en el usuario.

De esta forma, el modelo de seguridad de Android define un régimen de autorizaciones, destinadas a proteger los recursos y funciones del dispositivo, que requiere que cada aplicación solicite los permisos necesarios con implicaciones a nivel de seguridad o privacidad, como por ejemplo realizar una llamada, acceso a la libreta de direcciones o escribir en la memoria externa. Estos permisos se deben especificar cuando la aplicación se empaqueta en el archivo *AndroidManifest* y no se pueden modificar en tiempo de ejecución. Durante la instalación, se informa al usuario qué permisos solicita la aplicación. Si el usuario acepta los términos, se asume que todas las solicitudes de permisos están concedidas, no pudiéndose denegar solicitudes individuales.

Si un permiso necesario para una función no se solicita en el manifiesto, cuando la aplicación intente acceder a dicha función se genera una excepción. En el caso de un error silencioso, se añade un mensaje de error de permiso al registro de sistema de Android.

En la etiqueta *<uses-permission>* del archivo *AndroidManifest.xml* se identifican los distintos permisos que la aplicación solicita. Por ejemplo, la siguiente figura se muestra parte del archivo de manifiesto en el que se indican el permiso de escritura en memoria externa que la aplicación necesita:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.test.elma"
        android:versionCode="1"
        android:versionName="1.0" >
        <uses-sdk
                android:minSdkVersion="8"
                android:targetSdkVersion="15" />
        <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE">
        </uses-permission>
```

Figura 26. Solicitud de permisos de escritura en memoria externa en una aplicación

Los permisos pueden definirse de una forma más completa. El elemento *<uses-premission>* contempla una serie de atributos que definen y matizan el alcance del permiso dado:

- *name*: debe ser un nombre de alguno de los listados en la clase `android.Manifest.permission`
- *permissionGroup*: permite especificar a un grupo asociado el permiso. Los posibles grupos se encuentran listados en la clase `android.Manifest.permisson_group` y pueden tener valores como `ACCOUNTS` (cuentas válidas de Google), `COST_MONEY` (acciones vinculadas a un pago) o `PHONE_CALLS` (acciones relacionadas con llamadas).
- *protectionLevel*: determina el nivel de riesgo del permiso, y en función del

mismo, influye en cómo el sistema otorga o no el permiso a la aplicación. Oscila entre valores desde el 0 hasta el 3.

Al igual que se definen permisos de más alto nivel, también se pueden definir permisos específicos para un componente determinado o incluso definir permisos personalizados.

(3) Permisos con URIs

A pesar que cada aplicación se ejecuta bajo un identificador único, creándose una capa de aislamiento alrededor de cada proceso, el intercambio de datos entre aplicaciones es posible, pero debe ser explícito.

Cuando se implementa un proveedor de contenido, es necesario especificar el grado de protección de lectura y/o escritura sobre los datos que almacena, dado que esta información es accesible mediante una URI. Si esta dirección se pasa, por ejemplo, a un navegador, éste no debería ser capaz de acceder a dichos datos a no ser que estén concedidos los permisos por URI.

Los *Intent-filters* definen y delimitan qué tipos de *Intent* implícitos puede lanzar la *Activity*, y opcionalmente una URI que regularmente es usada por esta acción. Por ejemplo, cuando se inicie un *Intent* que no especifique a qué actividad va dirigido, el sistema buscará la aplicación que esté registrada y pueda responder a la necesidad, y ejecutará la acción. En caso de existir varias aplicaciones, se lanzará al usuario un cuadro de diálogo en el que pueda elegir la más conveniente.

La información que pasan los *intents* debe estar contenida en la definición del <*intent-filter*>, compuesta por tres campos:

Action: tipo de acción llevada a cabo. Pueden ser dadas por la clase *Intent*, por una API de Android o definidas por el desarrollador.

- *Data*: informa del identificador (URI) del dato que se asocia a la acción y del tipo de ese dato. Es importante la coherencia ya que si la acción requiere un dato de tipo texto, un *Intent* con un dato de tipo imagen no podrá ser lanzado.

- *Category*: información adicional sobre el tipo de componente al que va dirigido el *Intent*.

```
<activity class=".NotesList" android:label="@string/title_notes_list">
        <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
        <intent-filter>
                <action android:name="android.intent.action.VIEW" />
                <action android:name="android.intent.action.EDIT" />
                <action android:name="android.intent.action.PICK" />
                <category android:name="android.intent.category.DEFAULT" />
                <data android:mimeType="vnd.android.cursor.dir/vnd.google.note" />
        </intent-filter>
```

Figura 27. Esquema personalización URI

Firma de aplicaciones

Android requiere que las aplicaciones estén firmadas por sus desarrolladores. Con este certificado, cuya firma identifica al autor de la aplicación, se restringe la actualización de una versión que no sea publicada por el autor original.

No es necesario vincular a una autoridad de certificado, el único cometido del certificado es crear una relación de confianza entre las aplicaciones. Mediante la firma, la aplicación lleva adjunta su autoría.

3.5. API de seguridad de Java

El modelo de seguridad de Java ha ido evolucionando con las distintas versiones del entorno de desarrollo Java (JDK), pasando de un modelo inicial muy sencillo y restrictivo, a un modelo actual más complejo y flexible.

El conjunto de clases de seguridad distribuidas con Java 2 SDK pueden dividirse en dos subconjuntos:

1. Clases relacionadas con el control de acceso y la gestión de permisos
2. Clases relacionadas con la criptografía

El soporte que da el JDK a la criptografía se divide a su vez en dos grandes bloques, ambos incluidos en la API de Java:

1. JCA: Java Cryptography Architecture.

   Permite tener un sistema de autentificación fiable sobre el que implementar un sistema de control de acceso más flexible que el modelo de sandbox. Forma parte del *runtime* de la máquina virtual de Java a partir de la versión 1.2, bajo el paquete `java.security`, y no está sujeto a restricciones de exportación. El problema es que estas herramientas no son suficientes para el envío seguro de datos, porque carecen de algoritmos de cifrado.

2. *JCE: Java Cryptography Extension*.
   Dispone del soporte necesario para implementar criptografía de clave simétrica y asimétrica, así como para la generación y manipulación de las claves que estos emplean. Las clases de la librería JCE están bajo el nombre `javax.crypto.*`.

La separación entre el JCA y el JCE estuvo motivada por las reglas de exportación de software criptográfico en los EEUU. Las clases distribuidas con el JDK estándar solo proporcionaban las librerías de JCA, por lo que fabricantes de fuera de los EEUU

realizaron implementaciones de la librería JCE que distribuirían gratuitamente. Entre ellos destacan bouncycastle y cryptix.

A partir del JDK 1.4 se pudo exportar la librería JCE como parte de su máquina virtual, aunque los demás proveedores siguen resultando útiles ya que disponen de más algoritmos. (Talens-Oliag 1999)

La arquitectura de JCA y la de JCE se han diseñado alrededor de dos principios básicos:

1. Independencia e interoperabilidad de las implementaciones:

    - Las aplicaciones no necesitan implementar la seguridad por ellas mismas, sino que solicitan este servicio de la plataforma de Java, posible a través de una arquitectura basada en proveedores. Una aplicación puede confiar en múltiples proveedores independientes.

    - Los proveedores son interoperables a través de las aplicaciones, es decir, una aplicación no está obligada a un proveedor específico, lo mismo que un proveedor no está obligado a una aplicación específica.

2. Extensibilidad de los algoritmos:

    - Java soporta la instalación de proveedores personalizados que implementen nuevos estándares o servicios propietarios.

Ambos principios son complementarios. Por ejemplo, se pueden utilizar servicios criptográficos tales como la firma digital sin preocuparse sobre los detalles de implementación o los algoritmos que los implementan. Si no se desea una independencia de algoritmo, se permite el uso de una implementación específica de un proveedor. (Hernández, Seguridad, criptografía y comercio electrónico con Java 2007)

JCA

Las clases básicas de JCA están distribuidas en los siguientes paquetes: (Oracle n.d.)

`Java.security`

Consiste básicamente en clases abstractas e interfaces que encapsulan conceptos de seguridad como claves asimétricas, resúmenes de mensajes y firmas digitales. Algunas de estas clases son:

- KeyPairGenerator. Clase abstracta capaz de generar un par de claves asimétricas del algoritmo con el que se inicialice.

- PrivateKey y PublicKey. Interfaces para clave privada y pública, respectivamente, para el uso en algoritmos asimétricos.

- MessageDigest. Clase abstracta que proporciona la funcionalidad de algoritmos resumen.
- SecureRandom. Esta clase proporciona un generador de números pseudo aleatorios criptográficamente seguro.
- Signature. Clase abstracta capaz de generar y verificar firmas digitales.

`Java.security.cert`

Añade soporte para la gestión de certificados, listas de revocación (CRL) y rutas de certificación. Las clases más importantes incluidas en paquete son:

- Certificate. Clase abstracta para gestionar certificados.
- CertificateFactory. Se emplea para generar certificados y listas de revocación (CRL).
- CRL. Clase abstracta que representa las listas de revocación de certificados.
- X509Certificate. Clase abstracta que representa los certificados X.509.

`Java.security.interfaces`

Paquete de interfaces necesarias para generar claves asimétricas (pública y privada) específicas para los algoritmos RSA, DSA y EC.

- DSAKey, DSAKeyPairGenerator, DSAPrivateKey, DSAPublicKey
- ECKey, ECPrivateKey, ECPublicKey
- RSAKey, RSAPrivateKey, RSAPublicKey

`Java.security.spec`

Incluye las clases e interfaces necesarias para generar claves y parámetros específicos de los siguientes estándares soportados: DSA, RSA y EC.

JCE

Proporciona las clases e interfaces para la implementación de algoritmos de cifrado, descifrado o acuerdo de claves. Soporta tanto cifradores en bloque como de flujo, simétricos y simétricos. Se pueden integrar implementaciones de cifradores desde diferentes proveedores usando el SPI (*Service Provider Interface*). También soporta autenticación basada en MAC y HMAC.

Al igual que el JCA, el JCE emplea un modelo basado en el uso de proveedores. El paquete consta de un paquete principal, denominado `javax.crypto`, que consta de

clases que representan los conceptos de cifrado, acuerdos de claves y códigos de autentificación de mensajes, y sus clases de interfaz de proveedor (SPI).

Las clases más importantes son:

- Cipher. Clase que proporciona la implementación de algoritmos criptográficos. Se puede determinar el modo de funcionamiento del algoritmo y el tipo de relleno.
- KeyAgreement. Clase que proporciona la funcionalidad de un protocolo de acuerdo o intercambio de claves simétricas.
- KeyGenerator. Clase que genera claves simétricas.
- Mac. Clase que proporciona la funcionalidad de las funciones MAC.
- SecretKey. Interfaz que representa la clave secreta del algoritmo simétrico implementado.

`Javax.crypto.interfaces`

Proporciona las interfaces necesarias para implementar DH especificado en PKCS#3:

- DHKey, DHPrivateKey, DHPublicKey

`Javax.crypto.spec`

Proporciona clases e interfaces que especifican tanto los parámetros como las claves necesarias para los algoritmos de cifrado soportados (PKCS#3 DH, FIPS-46-2 DES y PKCS#5 PBE). Las especificaciones de claves son representaciones transparentes de las claves. Las claves se pueden especificar de un modo determinado del algoritmo o en un formato de codificación independiente del algoritmo, como el ASN.1.

Capítulo 4

Estudio de Eficiencia de Algoritmos de Cifrado

Uno de los objetivos de este proyecto es analizar el comportamiento de los algoritmos de seguridad simétricos para el cifrado y descifrado de datos en términos de coste temporal.

4.1. Algoritmos analizados y parámetros de estudio

El propósito de este capítulo es realizar un estudio de la eficiencia en términos de coste temporal de los algoritmos de cifrado soportados en Android.

Para ello, se ha desarrollado una aplicación en la que se ha implementado un cifrador que hará posible la toma de las medidas necesarias para la consecución de este propósito. Ha sido necesario un estudio previo de los algoritmos de cifrado disponibles en esta plataforma. Dado que las aplicaciones en Android están escritas en Java, se ha hecho uso del paquete `javax.crypto`, que proporciona las clases e interfaces necesarias para realizar este estudio.

Una vez conocidas las opciones disponibles, se procedió a implementar la aplicación, y realizar las medidas para el estudio. Al ejecutarse la aplicación, el usuario puede seleccionar el algoritmo de cifrado (DES, 3DES y AES) junto con los parámetros que se ajusten a las necesidades del estudio. Estos parámetros son configurables, y permiten la variación del tamaño de la clave (no disponible para DES, ya que por definición solo soporta un único tamaño de clave), el modo de operación (ECB, CBC, CFB y OFB), así como el tamaño de fichero a cifrar.

No obstante, el funcionamiento de la aplicación es muy sencillo. Una vez seleccionados los parámetros de estudio, se llama al generador de clave que inicializará una clave del tamaño seleccionado. Al cifrador se le pasa esta clave y el archivo original, alojado en la memoria interna del dispositivo, y se le inicializa con un determinado modo de operación, para comenzar el proceso de cifrado. A su finalización, se genera un archivo cifrado, que se habrá grabado en la memoria externa del dispositivo. Con este fichero encriptado, y la clave anteriormente generada, el cifrador procede a descifrarlo, recuperando la información contenida en el archivo original.

Aunque en la documentación del API, Java[1] soporta también los algoritmos RC2 y RC5, no estaban incluidos en ninguno de los proveedores criptográficos (Crypto, BC y HarmonyJSSE) disponibles por defecto, por lo que no se pudo realizar el estudio de éstos.

Tras la toma de medidas, se procesan los datos obtenidos, para la realización de una comparativa del tiempo de procesamiento entre los diferentes algoritmos, frente a la variación de los distintos parámetros seleccionables, comunes para todos los algoritmos, tales como el modo de funcionamiento y tamaño del fichero, y específicos, como el tamaño de la clave. En la siguiente tabla se puede observar los tamaños de clave estudiados para cada algoritmo:

[1] http://developer.android.com/reference/java/security/spec/AlgorithmParameterSpec.html

Tabla 9. Parámetros variables algoritmos

Algoritmo	Tamaño de clave bits
DES	56
3DES	168, 112
AES	256, 192, 128

Para cada configuración, se han tomado medidas de hasta 20 repeticiones, de las cuales se ha seleccionado una muestra con los 10 resultados más óptimos que se ajustasen a un intervalo de confianza de al menos un 95%. Con esa muestra, se ha calculado la media y la desviación que representará al algoritmo.

Para poder efectuar un análisis completo, se ha procedido a realizar un estudio de cada algoritmo por separado, para finalizar con una comparativa de todos ellos.

En los gráficos que se van a mostrar a continuación solo se muestran los tiempos de cifrado ya que se entiende que los tiempos de descifrado toman valores similares al efectuar la misma acción aunque en sentido contrario.

Para poder efectuar un análisis completo, se ha procedido a realizar un estudio de cada algoritmo por separado, para finalizar con una comparativa de todos ellos.

A continuación se muestra el orden que se seguirá en la presentación de los resultados:

- Algoritmo DES
- Algoritmo 3DES
- Algoritmo AES
- Comparativa eficiencia tiempo de generación de la clave secreta
- Resumen y discusión

4.2. Algoritmo DES

En este estudio, se ha dejado fijo el tamaño de la clave del algoritmo, DES con una clave de 56 bits, y se han ido variando tanto el modo de operación como el tamaño del fichero.

A continuación se representan los resultados obtenidos en forma gráfica en la siguiente Figura 28:

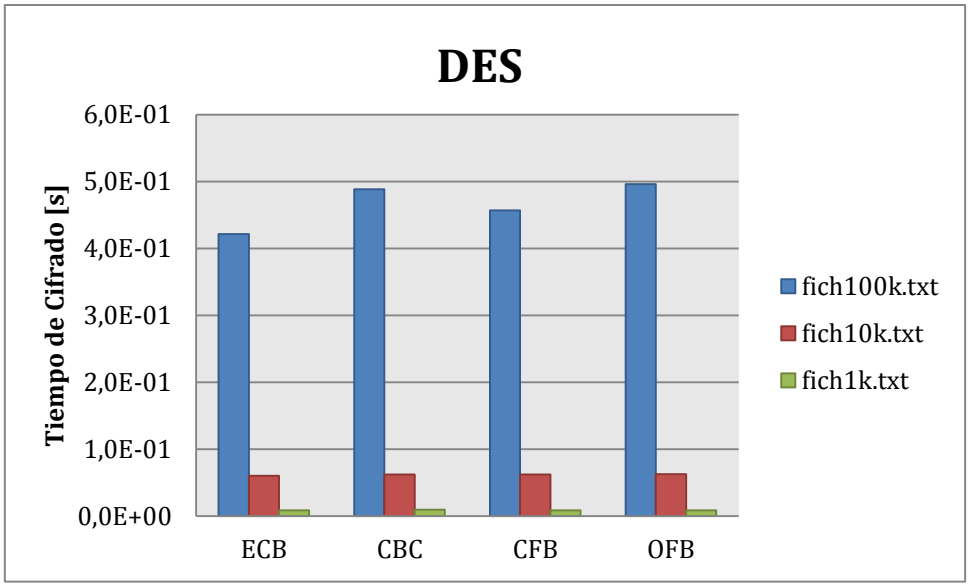


Figura 28. Evolución tiempo cifrado en DES

A partir de los resultados obtenidos, se observa una relación directa entre el tiempo de procesamiento con respecto al tamaño del fichero de entrada. El tiempo de cifrado es proporcional al tamaño del fichero; cuanto mayor es el volumen de datos a procesar, mayor es el tiempo necesario.

También se observa que el orden de magnitud temporal del algoritmo es muy similar independientemente del modo de funcionamiento del mismo, aunque en detalle se observarán diferencias. Destacan los modos ECB ser el más rápido y en el caso contrario, el modo de operación OFB es el más lento, aunque este comportamiento se observará con mayor detalle en el siguiente punto.

Debido a las diferencias en los órdenes de magnitud de los tiempos resultantes, se ha decidido representar las medidas obtenidas en tres gráficos diferenciados, uno para cada tamaño del fichero, véase Figura 29 a Figura 31.

Si el tamaño del fichero es el más grande, véase Figura 29, el modo de operación ECB es el más rápido y el OFB el más lento, en una diferencia de 74.64 ms, que constituye un 15% más de computación temporal.

La media temporal del algoritmo DES para un tamaño de fichero de 100KB es de 465.42 ms, situándose en mínimos de 421 ms en modo ECB y máximos de 495 ms en modo OFB.

Fijando el parámetro de fichero al tamaño intermedio, véase Figura 30, de nuevo el modo de operación ECB es más rápido y el OFB el más lento, en una diferencia de 2.54 ms, que supone un 4% más de computación temporal.

La media temporal del algoritmo DES para un tamaño de fichero de 10KB es de 61.69 ms, situándose en mínimos de 59.94 ms en modo ECB y máximos de 62.48 ms en modo

OFB. Se observa que a excepción del modo ECB, los otros tres modos tienen comportamientos temporales muy similares.

Fijando el parámetro de fichero de menor tamaño, véase Figura 31, el modo de operación CBC es el más lento, y el ECB sigue siendo el más rápido, aunque el resto de modos tienen tiempos muy próximos. La diferencia entre los extremos alcanza 1.07 ms, que constituye un 11.5% más de computación temporal.

La media temporal del algoritmo DES para un tamaño de fichero de 1KB es de 8.56 ms, situándose en mínimos de 8.21 ms en modo ECB y máximos de 9.28 ms en modo CBC.

Como ya se ha venido observando, la carga temporal está directamente ligada con el tamaño del archivo a cifrar; es decir, cuanto mayor es el volumen de datos a cifrar, mayor es el tiempo de cifrado.

Con respecto a los modos de operación, hay una diferencia computacional entre el más rápido y el más lento que varía entre un 4% y un 15%, pero que se traduce en órdenes de magnitud a 3ms y 75ms, una diferencia más acusada a mayor volumen de datos a cifrar.

El modo ECB ha destacado por ser el más rápido en todas las condiciones. El tiempo necesario para cifrar 1KB es 7.3 veces el necesario para cifrar un fichero 10 veces mayor. En cambio, el tiempo necesario para cifrar 10KB es 7.02 veces el necesario para cifrar un fichero 10 veces mayor.

Por el contrario, si el algoritmo DES se configura en modo OFB, presenta los tiempos mayores, por tanto, es la combinación más lenta. El tiempo necesario para cifrar 1KB es 8.2 veces el necesario para cifrar un fichero 10 veces mayor. En cambio, el tiempo necesario para cifrar 10KB es 7.9 veces el necesario para cifrar un fichero 10 veces mayor.

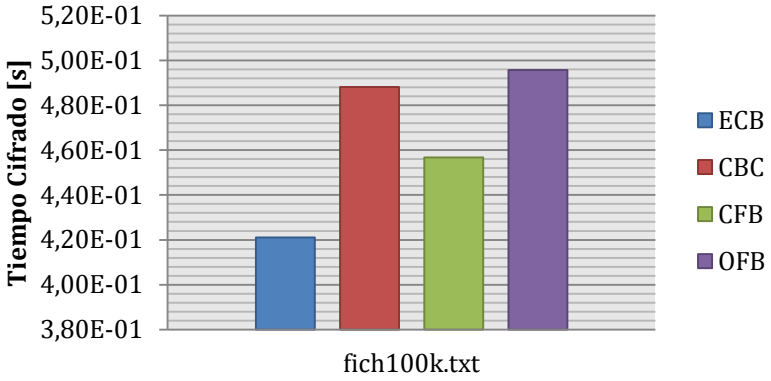


Figura 29. Evolución tiempo cifrado DES fichero 100KB

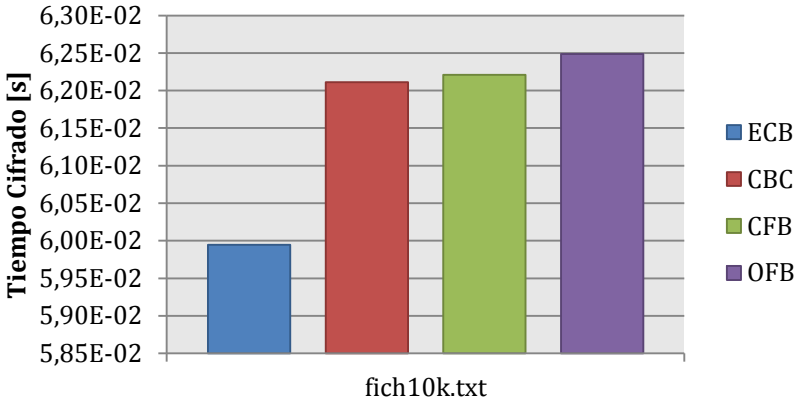


Figura 30. Evolución tiempo cifrado DES fichero 10KB

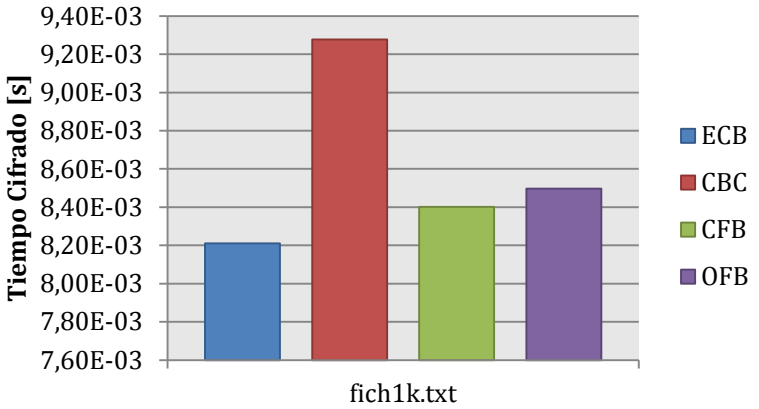


Figura 31. Evolución tiempo cifrado DES fichero 1KB

Tabla 10. Resumen resultados obtenidos DES

CLAVE	FICHERO	MODO RÁPIDO	MODO LENTO	DIFERENCIA	PROMEDIO
56	100KB	ECB	OFB	15%	465.42ms
56	10KB	ECB	OFB	4%	61.69 ms
56	1KB	ECB	CBC	11.5%	8.59 s

4.3. Algoritmo 3DES

En este estudio, se ha podido variar tanto el tamaño de la clave del algoritmo 3DES, con dos posibles opciones 112 y 168 bits, así como el modo de operación y el tamaño del fichero.

A continuación se representan los resultados obtenidos en forma gráfica en la siguiente Figura 32:

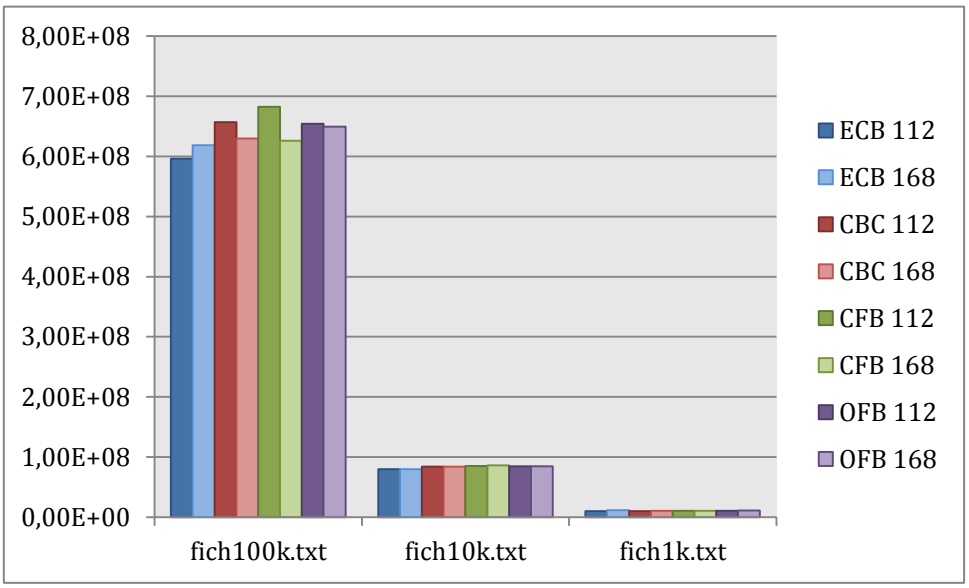


Figura 32. Evolución tiempo cifrado 3DES

Como era de esperar, hay una relación directa entre el tiempo de procesamiento con respecto al tamaño del fichero de entrada: cuanto mayor es el fichero, más tiempo de cifrado es necesario.

Debido a las diferencias en los órdenes de magnitud de los tiempos resultantes, se ha decidido representar las medidas obtenidas en tres gráficos diferenciados, uno para cada tamaño del fichero, véase Figura 33 a Figura 35.

Si evaluamos el comportamiento del algoritmo 3DES sobre un fichero de gran tamaño, véase Figura 33, el modo de cifrado más rápido es ECB y el más lento sería CFB para los dos tamaños de la clave.

El modo ECB es el más rápido, con valores resultantes de 535ms y 597ms para claves de 112bits y 168bits respectivamente. Coincide también el modo más lento, CFB, con resultados obtenidos de 682ms y 644ms, que se observa como el tiempo es superior aunque la clave sea de menor tamaño. La diferencia temporal entre usar uno u otro modo se traduce en 147.45ms y 47.3ms, que constituye un 21.6% y 7.3% de diferencia computacional.

La media temporal del algoritmo 3DES para una clave de 112bits en estas condiciones es de 625 ms, y 628 ms si la clave es de 168bits, diferenciándose en 3 ms aumentando el tamaño de la clave.

Si el fichero seleccionado es de tamaño mediano, véase Figura 34, vuelve a coincidir el modo de cifrado más rápido, ECB, y el más lento, CFB, para los dos tamaños de clave.

Con el modo más rápido, ECB, se obtienen resultados de 79.76ms y 81.34 ms para claves de 112bits y 168bits respectivamente. Con el modo más lento, CFB, se obtienen tiempos de 86.46ms y 86.65ms. Se observa como el diferencial entre ambos modos es del orden de decimas de ms. Entre usar el modo más rápido y más lento la diferencia asciende a 6.7ms y 5.3 ms, que constituyen un 7.7% y 6.12% de diferencia computacional.

La media temporal del algoritmo 3DES para una clave de 112bits y un tamaño de fichero 10KB es de 84.01 ms, y de 84.5 ms si la clave es de 168bits, apenas 0.47 ms superior con respecto a la clave inferior.

Cuando el volumen de datos es el más pequeño, véase Figura 35, se sigue observando el mismo comportamiento que en los anteriores modos: ECB es el más rápido y CFB seguido de OFB los más lentos.

Fijando el parámetro de clave a 112, el modo de operación ECB es el más rápido con 10.15 ms y el CFB el más lento con 10.89 ms, en una diferencia de 0.74 ms, que constituye un 6.81% más de computación temporal. Para el caso de un tamaño de clave de 168, el modo más rápido es CBC con 10.91 ms, y el más lento es ECB con 12.03 ms, estableciendo una diferencia temporal de 1.21 ms, lo que determina un 9.32%.

Bajo estas observaciones, el tiempo de procesamiento de un fichero de 1K es aproximadamente 7.6 veces lo que se necesitaría si el fichero es de 10K. Esta relación es aproximadamente la misma de un fichero 10K a 100K, por lo que el coste temporal de calcular 1K de datos es 57.76 veces lo que se tardaría si los datos son 100K.

En todos los casos, los resultados de utilizar una clave de mayor tamaño son superiores a usar una clave de tamaño menor, no obstante estas diferencias temporales son, en promedio 0.83 ms. Esta conclusión, se puede extrapolar a un término genérico, es decir, el uso de una clave de mayor tamaño en no se traduce en un aumento de coste computacional considerable. No existen diferencias significativas en función de la clave, es decir, según los resultados, el tiempo en cifrado es del mismo orden independientemente del tamaño de ésta. Utilizar una clave de mayor tamaño no supone un mayor coste temporal, y en cambio, si está aportando mayor seguridad frente a un ataque.

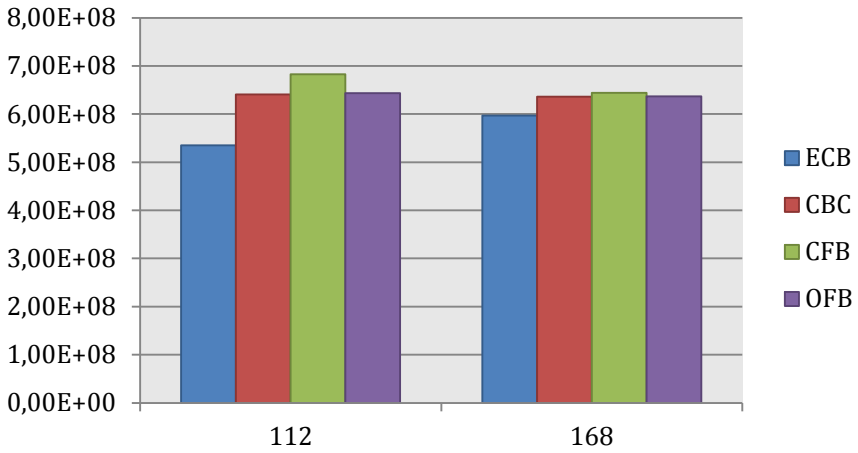


Figura 33. Evolución tiempo cifrado 3DES fichero 100KB

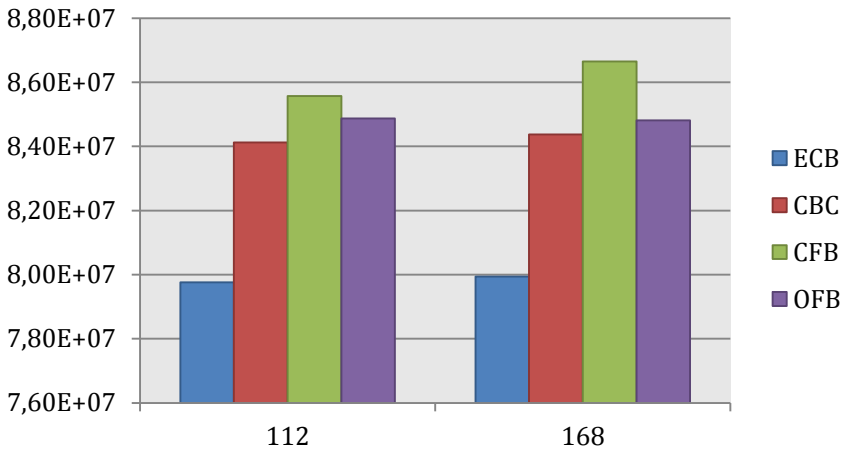


Figura 34. Evolución tiempo cifrado 3DES fichero 10KB

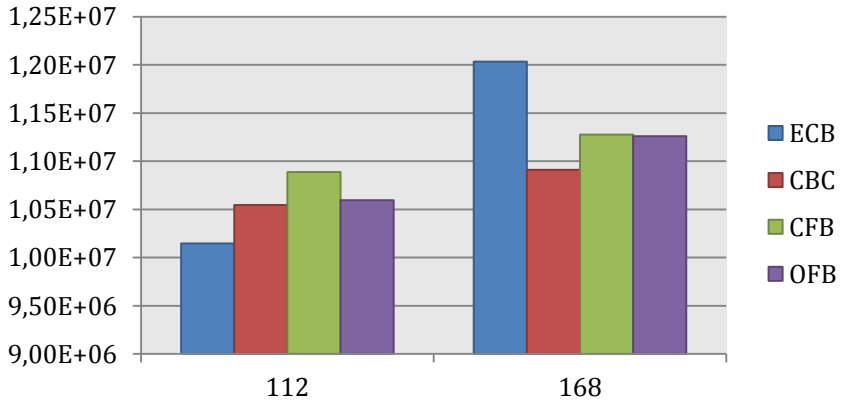


Figura 35. Evolución tiempo cifrado 3DES fichero 1KB

Tabla 11. Resumen resultados obtenidos 3DES

CLAVE	FICHERO	MODO RÁPIDO	MODO LENTO	DIFERENCIA	PROMEDIO
112	100KB	ECB	CFB	21.6%	625.50 ms
168	100KB	ECB	OFB	7.3%	628.55 ms
112	10KB	ECB	CFB	7.75%	84.01 ms
168	10KB	ECB	CFB	6.12%	84.53 ms
112	1KB	ECB	CFB	6.81%	10.54 ms
168	1KB	CBC	ECB	9.32%	11.37 ms

4.4. Algoritmo AES

El algoritmo AES puede funcionar con tres tamaños diferentes de clave: 128, 192 y 256 bits. Por tanto, en este estudio, se ha podido variar tanto el tamaño de la clave, así como el modo de operación y el tamaño del fichero.

A continuación se representan los resultados obtenidos en forma gráfica en la siguiente Figura 36:

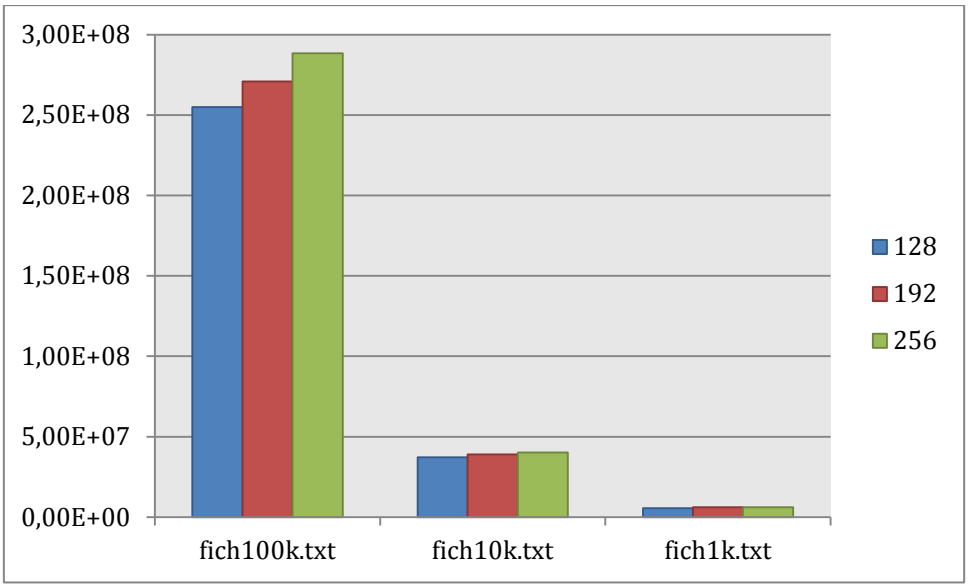


Figura 36. Evolución tiempo cifrado AES modo ECB

Como se viene observado en el resto de algoritmos, el tiempo de cifrado aumenta con el volumen de datos.

Si se fija el parámetro del fichero al tamaño grande, véase Figura 37, el modo de cifrado más rápido es ECB para los tres tamaños de la clave, en cambio no hay una unanimidad para el modo más lento.

Se cumple que a mayor tamaño de la clave, el tiempo de cifrado aumenta, aunque esta diferencia no tiene la misma proporción que la clave.

Fijando el parámetro de clave a 128bits, el modo de operación ECB es el más rápido con 255 ms y el CBC el más lento con 332 ms, diferenciándose en 76.58 ms, que constituye un 23.09 % más de computación temporal. Para el caso de un tamaño de clave de 192bits, el modo más rápido también es ECB con 271 ms, y el más lento CFB con 305 ms, estableciendo una diferencia temporal de 34.6 ms, lo que constituye un porcentaje de 11.33%; para el caso del tamaño de clave más grande, 256bits, ECB sigue siendo el más rápido, con 288 ms y el más lento CFB con 316 ms, existiendo una diferencia de 8.63%. Se observa que según va aumentando el tamaño de la clave, la diferencia temporal entre los modos de operación va disminuyendo.

La media temporal del algoritmo AES para una clave de 128bits y un tamaño de fichero 100KB es de 292.39 ms; si la clave es de 168bits es de 294.78 ms y si la clave son 256bits un promedio de 305.75ms, unos 13.36 ms (4%) superior con respecto a la clave más pequeña.

Si el tamaño fijado es el mediano, véase Figura 38, coinciden el modo de cifrado más rápido es ECB y el más lento CFB para los tres tamaños de la clave.

Se cumple que a mayor tamaño de la clave, el tiempo de cifrado aumenta, aunque esta diferencia no tiene la misma proporcionalidad de la clave.

Fijando el parámetro de clave a 128bits, el modo de operación ECB tarda 37.13 ms y el CFB 41.41 ms, diferenciándose en 4.28 ms, que constituye un 10.34% más de computación temporal. Para el caso de un tamaño de clave de 192, con ECB tarda 39.05 ms, y CFB 43.56 ms, estableciendo una diferencia temporal de 4.51 ms, lo que constituye un porcentaje de 10.35%; para el caso del tamaño de clave más grande, ECB cifra en 40.10 ms y el CFB en 46.23 ms, existiendo una diferencia de 6.13 ms, que en porcentaje asciende a 13.25%.

La media temporal del algoritmo AES para una clave de 128b y un tamaño de fichero 10KB es de 39.66 ms; si la clave es de 192b es de 41.60 ms y si la clave son 256b un promedio de 43.69ms, diferenciándose en solo unos 4 ms superior con respecto a la clave más pequeña.

Si el tamaño del fichero es el más pequeño, véase Figura 39, el modo de cifrado más rápido es ECB para los tres tamaños de la clave, en cambio el más lento no es unánime.

Fijando el parámetro de clave a 128, el modo de operación ECB tarda 5.61 ms y el más lento, el modo OFB necesita 6.14 ms, diferenciándose en 0.56 ms, que constituye un 9.05% más de computación temporal. Para el caso de un tamaño de clave de 192, con ECB tarda 6.16 ms, y el modo CFB 6.41 ms, estableciendo una diferencia temporal entre ambos extremos de 0.26 ms, lo que constituye un porcentaje de 4.01%; si la clave es la de tamaño mayor, ECB cifra en 6.16 ms y el CBC en 7.17 ms, existiendo la mayor diferencia de 1.01 ms, que en porcentaje asciende a 14.06%.

La media temporal del algoritmo AES para una clave de 128b y un tamaño de fichero 1KB es de 5.97 ms; si la clave es de 192b es de 6.27 ms y si la clave son 256b un promedio de 6.68 ms, unos 0.71 ms superior con respecto a la clave más pequeña.

Como en los casos anteriores, a mayor tamaño de la clave, mayor tiempo de cifrado.

La diferencia entre utilizar un modo de operación u otro puede variar entre un 4% y un 21% de tiempo dependiendo de si se utiliza el más rápido o el más lento. El modo más rápido es siempre el ECB, bastante lógico puesto que no utiliza realimentación, y es el más simple de todos al ir cifrando bloque a bloque. Haciendo una generalización, el modo más lento es el CFB.

Como conclusión se observa que a pesar de que el uso de la clave de mayor tamaño, 256b, es el doble que la clave más pequeña, 128b, el tiempo de cifrado solo llega a aumentar en un factor 1.1, prácticamente despreciable teniendo en cuenta el aumento en el nivel de seguridad.

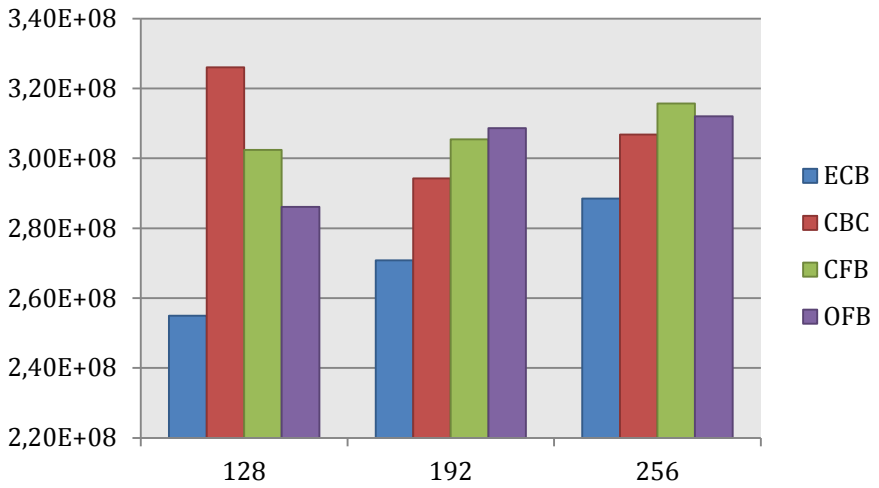


Figura 37. Evolución tiempo cifrado AES fichero 100KB

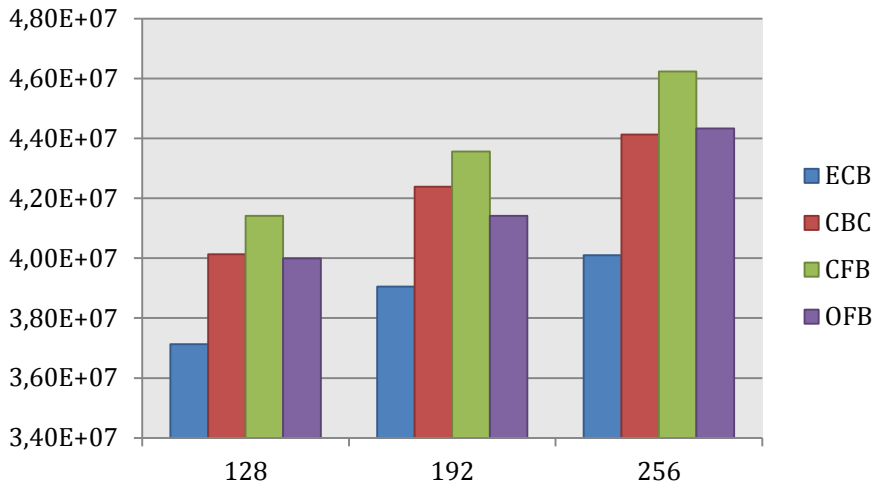


Figura 38. Evolución tiempo cifrado AES fichero 10KB

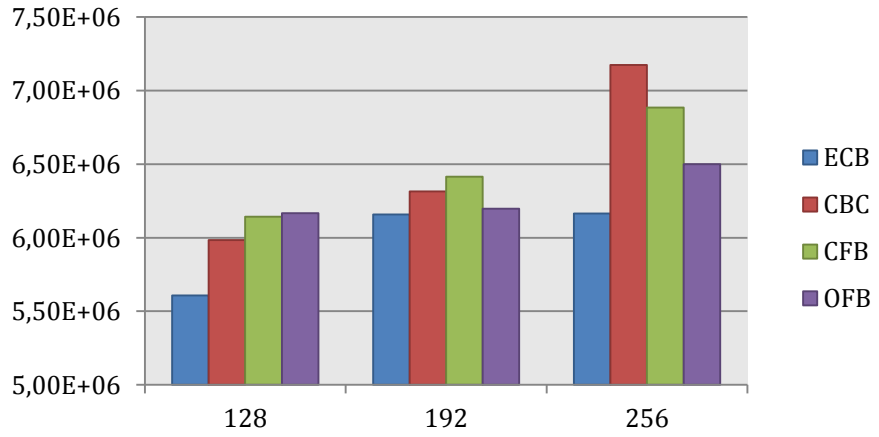


Figura 39. Evolución tiempo cifrado AES fichero 1KB

Tabla 12. Análisis resultados AES

Clave	Fichero	Modo Rápido	Modo Lento	Diferencia	Promedio
128	100KB	ECB	CBC	23.09%	292.39 ms
192	100KB	ECB	CFB	11.33%	294.78 ms
256	100KB	ECB	CFB	8.63%	305.75 ms
128	10KB	ECB	CFB	10.34%	39.66 ms
192	10KB	ECB	CFB	10.35%	41.60 ms
256	10KB	ECB	CFB	13.25%	43.69 ms
128	1KB	ECB	OFB	9.05%	5.97 ms
192	1KB	ECB	CFB	4.01%	6.27 ms
256	1KB	ECB	CBC	14.06%	6.68 ms

4.5. Resumen y Discusión

Evolución del tiempo de generación de la clave secreta

El tiempo de generación de claves secreta solo dependerá del algoritmo seleccionado y del tamaño de la clave, siendo independiente del resto de parámetros, como el modo de operación.

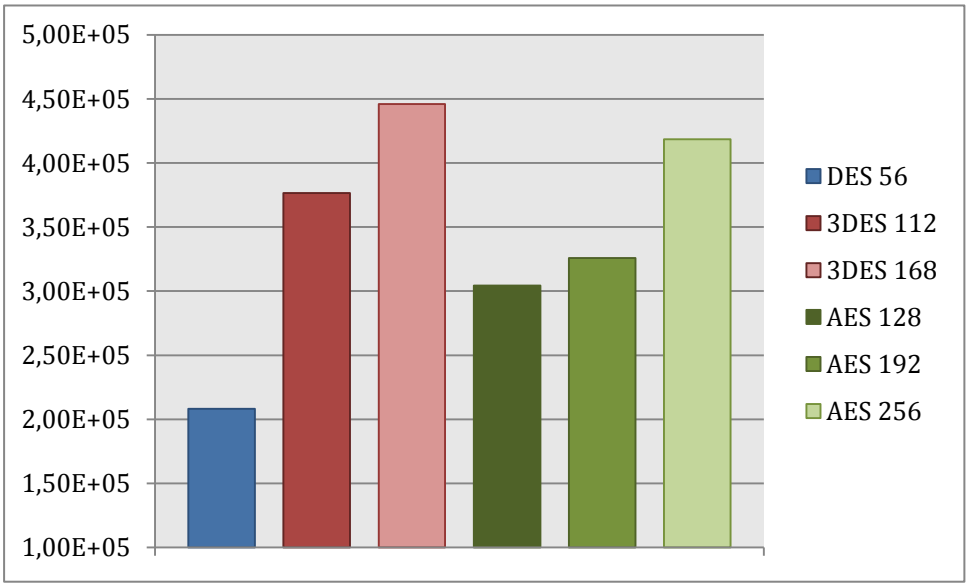


Figura 40. Evolución tiempo generación clave algoritmos simétricos

Según la Figura 40 el algoritmo que tarda menos en generar su clave es el DES, que coincide con el que posee menor tamaño de clave, 56bits. Previsiblemente, si la clave es más grande, el tiempo de generación es superior. En cambio, el algoritmo 3DES obtiene peores resultados con respecto a AES a pesar de tener tamaños de clave inferiores. Así mismo, en el algoritmo AES se observa que el tiempo de generación de las claves de tamaño 128bits y 192 bits son muy próximos, diferenciándose en 21 ms siendo 1.5 el aumento de la seguridad de la clave.

Tabla 13. Resumen tiempos generación clave secreta

ALGORITMO	CLAVE	PROMEDIO	DESVIACIÓN
DES	56	208.277,7917	2.290,0570
3DES	112	376.527,6667	7.015,6127
3DES	168	445.972,1667	8.240,6717
AES	128	304.305,6667	9.360,1203
AES	192	325.833,4167	6.009,2059
AES	256	418.472,1667	6.374,6565

Este estudio se va a subdividir para cada tamaño de fichero para facilitar su análisis, aunque solo se va a mostrar la comparativa gráfica para un tamaño de fichero seleccionado.

La primera conclusión que se obtiene al observar la comparativa de todos los algoritmos, véase Figura 41 a Figura , para cualquier tamaño de archivo, es que AES es el más rápido y 3DES el más lento.

En la Tabla 14 se representa la diferencia tanto temporal como porcentual entre el algoritmo con los resultados más rápidos y el más lento, para diferentes tamaños de fichero y para cada modo de configuración:

Tabla 14. Comparativa algoritmos simétricos

Tamaño Fichero	Modo	Algoritmo Rápido	Algoritmo Lento	Diferencia Temporal	Diferencia Porcentual	Factor (t_L/t_R)
100KB	ECB	AES 128	3DES 168	363,516 ms	58.77%	2.43
100KB	CBC	AES 192	3DES 112	362,349 ms	55.19%	2.23
100KB	CFB	AES 128	3DES 112	380,074 ms	55.69%	2.26
100KB	OFB	AES 128	3DES 112	368,151 ms	56.27%	2.29
10KB	ECB	AES 128	3DES 168	42,804 ms	53.54%	2.15
10KB	CBC	AES 128	3DES 168	44,238 ms	52.43%	2.10
10KB	CFB	AES 128	3DES 168	45,238 ms	52.20%	2.09
10KB	OFB	AES 128	3DES 112	44,882 ms	52.88%	2.12
1KB	ECB	AES 128	3DES 168	6,425 ms	53.39%	2.15
1KB	CBC	AES 128	3DES 168	4,927 ms	45.15%	1.82
1KB	CFB	AES 128	3DES 168	5,134 ms	45.53%	1.84
1KB	OFB	AES 128	3DES 168	5,092 ms	45.23 %	1.83

En esta tabla se destaca cuál ha sido el algoritmo más rápido y más lento, para determinado tamaño de fichero y modo, se ha calculado la diferencia temporal entre los dos resultados, y se ha calculado qué porcentaje representa esa diferencia sobre el algoritmo más lento, así como cuantas veces el algoritmo más lento es mayor que el algoritmo más rápido.

Para un archivo de tamaño 100KB, la mayor diferencia porcentual se encuentra siendo AES128 algoritmo más rápido, y 3DES168 como el algoritmo más lento, para el modo ECB, superando el 58.77%. El resto de modos tienen un porcentaje algo inferior, aunque en todos los casos superan el 55%.

Se observa como esta diferencia se va haciendo más pequeña según el archivo va disminuyendo; de forma que la mayor diferencia porcentual, bajo las mismas condiciones, y para un fichero de 10KB es de 53.54% y si el fichero de 1KB, alcanza un 53.39%.

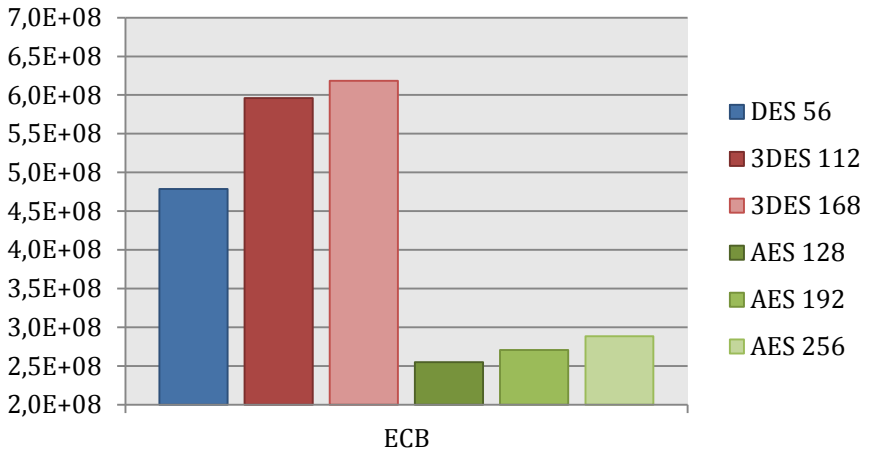


Figura 41. Evolución cifrado simétrico modo ECB fichero 100KB

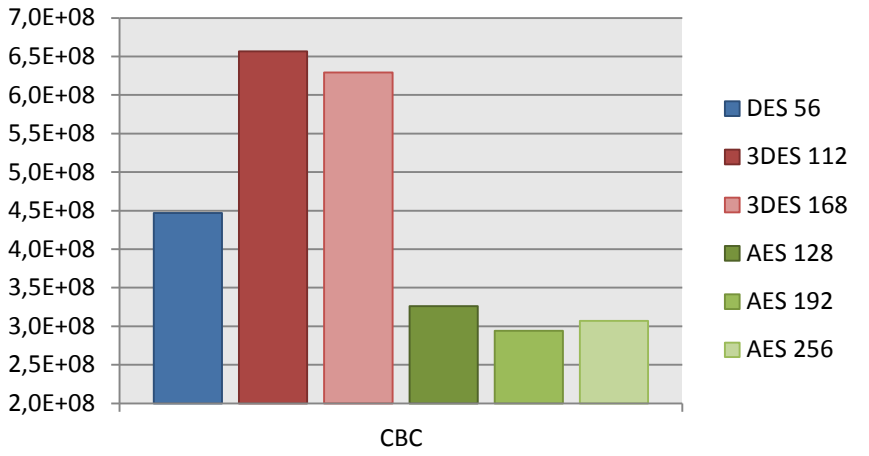


Figura 42. Evolución cifrado simétrico modo CBC fichero 100KB

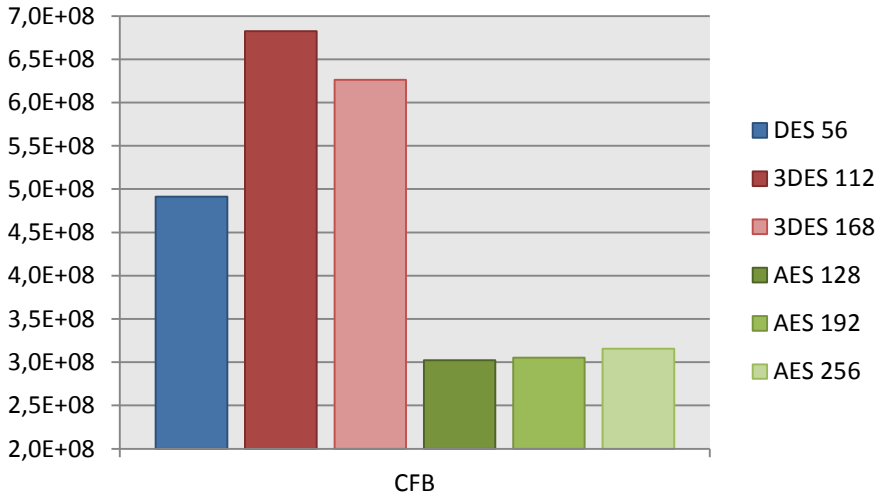


Figura 43. Evolución cifrado simétrico modo CFB fichero 100KB

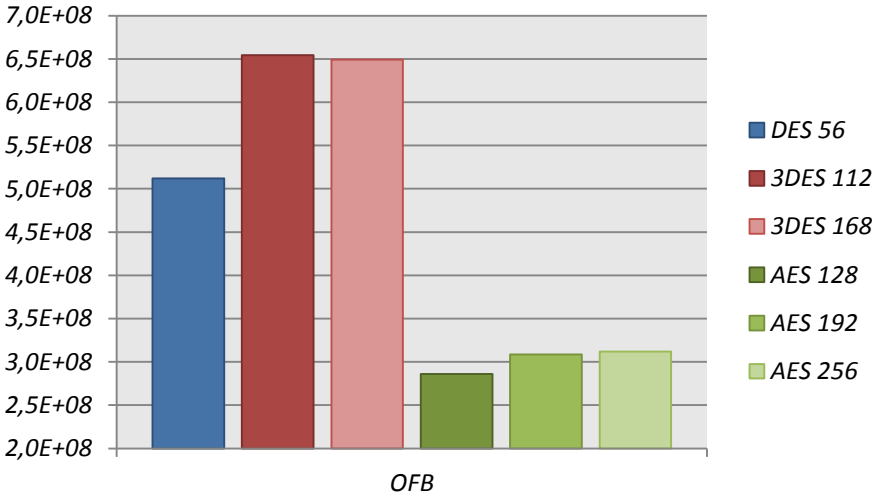


Figura 44. Evolución cifrado simétrico modo OFB fichero 100KB

O expresado como factor, el algoritmo 3DES168 puede ser hasta 2.43 veces más lento que AES128, en el caso de un fichero de 100KB, o esta diferencia se reduce hasta un factor 2.15 en el caso de un fichero de 1KB. Este resultado se observa para el modo de operación ECB, que siempre ha respondido por ser la configuración más rápida, y en la comparativa ha resultado ser el modo de operación con mayores diferencias entre los algoritmos.

Como ya se había observado en el estudio individual de cada algoritmo, el modo ECB es el más rápido, y generalmente el más lento varía entre CFB o OFB en función del tamaño del archivo, aunque las diferencias entre el resto de modos son del orden de décimas o centésimas de ms.

A partir de los resultados del algoritmo AES, se observa que a pesar que el tamaño de la clave se duplica (pasa de un tamaño 128bits hasta 256bits), el tiempo de cifrado no sigue ese factor, sino que la diferencia temporal entre ambos resultados toma un factor de 1.13 a 1.09, un valor tan pequeño que podría afirmarse que usar una clave más robusta no implica un mayor coste temporal ni computacional.

Así mismo, se observa un aumento en el tiempo de cifrado en el algoritmo AES de forma escalada con respecto al tamaño de la clave. Esta diferencia puede alcanzar décimas de ms.

El algoritmo DES, a pesar de ser el que menor tamaño de clave utiliza, es más lento que AES con su mayor tamaño de clave, en un factor que va desde 1.65 si el fichero es de 100KB a 1.36 si el fichero es de 1KB.

El algoritmo 3DES el más lento. Para tamaños de fichero de 100KB utilizar una clave de mayor tamaño obtiene mejores resultados que si se utiliza una clave de menor tamaño dado que apenas se percibe una gran diferencia computacional entre el uso de un tamaño de clave u otra. No obstante, esta diferencia sí se destaca más si el fichero es más pequeño.

Comparativa del tiempo de cifrado con respecto al tamaño del fichero de entrada.

De todo este estudio, se puede concluir que a mayor tamaño de fichero a cifrar, mayor es el tiempo necesario para realizar dicha operación.

Tabla 15. Comparativa algoritmos

FICHERO	AES 128	3DES 168	DIFERENCIA TEMPORAL	DIFERENCIA PERCENTUAL	MAYOR QUE
Pequeño	5,9753 ms	11,3701 ms	5,3948 ms	47,45%	1,90
Mediano	39,6651 ms	83,9415 ms	44,2763 ms	52,75%	2,12
Grande	292,3895 ms	630,7826 ms	338,2931 ms	53,65%	2,15

El porcentaje de diferencia entre el algoritmo más rápido con respecto al más lento está alrededor del 50%, es decir, que el algoritmo más rápido realiza la misma operación en la mitad de tiempo que el algoritmo más lento. Esta relación de velocidad va decreciendo según el volumen de datos es inferior.

Tras realizar un análisis exhaustivo sobre los datos obtenidos en este estudio, se puede concluir lo siguiente:

- Se ha demostrado como se produce un incremento del tiempo de cifrado proporcional al aumento del tamaño del fichero;
- El algoritmo más lento es 3DES y el más rápido AES. Por tanto, AES será el algoritmo que habrá que elegir en cualquier caso, ya que además, proporciona un mayor nivel de seguridad.
- El modo de operación más rápido para cualquier algoritmo ha sido ECB, entendible dado que no requiere de retroalimentación para procesar el cifrado, aunque de otra forma, es el más vulnerable. No hay una unanimidad para el modo de operación más lento, por lo que habrá que estudiar cuál es la casuística de ese momento para la elección del modo más adecuado.

Capítulo 5

Estudio de eficiencia algoritmos de firma y verificación

El objetivo de este estudio es analizar la eficiencia de los algoritmos de seguridad asimétricos usados en la firma y verificación de datos, en términos de coste temporal.

5.1. Algoritmos analizados y parámetros de estudio

El propósito de este capítulo es realizar un estudio de la eficiencia en términos de coste temporal de los algoritmos asimétricos de firma y verificación soportados en Android.

El funcionamiento de la aplicación es muy sencillo. Desde el menú principal el usuario selecciona el estudio de algoritmos asimétricos especificando sobre qué algoritmo desea realizar el estudio, siendo los algoritmos disponibles RSA y DSA. Posteriormente elige los parámetros sobre los que tomar las medidas, definiendo el tamaño del fichero (desde 1KB, 10 KB a 100KB), el tamaño de la clave, que variará en función del algoritmo seleccionado y la función hash que realizará el resumen del fichero seleccionado previamente. En la siguiente tabla se pueden observar los siguientes parámetros ajustables de cada algoritmo:

Tabla 16. Parámetros variables algoritmos asimétricos

Algoritmo	Tamaño de clave bits	Función Hash
RSA	4096, 2048, 1024, 512	MD4, MD5, SHA1, SHA224, SHA256, SHA384, SHA512
DSA	1024, 768, 512	SHA1, sin hash

Tras la toma de medidas, cubriendo todos los escenarios posibles, se procesan los datos obtenidos, de forma que se seleccionan los 10 resultados más óptimos que se ajusten a un intervalo de confianza del 95%.

En la aplicación se ha implementado el servicio Signature de Java, para la creación y verificación de firmas digitales, distribuidas en el paquete java.security. La clase KeyPairGenerator genera una pareja de claves del tamaño y algoritmo asíncrono seleccionado (RSA o DSA), KeyPair, donde se encapsulan una clave privada y una clave pública, ambas interfaces derivadas de Key. Con la clave privada y el archivo original, alojado en la memoria interna del dispositivo, comienza el proceso de firma. Esto supone realizar un resumen del mensaje original, con la función hash seleccionada por usuario, y cifrar dicho resumen con la clave privada del algoritmo seleccionado, considerándose el resultado final una firma digital. A su finalización, se genera un archivo firmado (la firma digital), grabado en la memoria externa del dispositivo. Para el proceso de verificación se toma de nuevo el archivo original, se le aplica la misma función hash y se codifica con el algoritmo seleccionado pero esta vez aplicando la clave pública. Si la firma original, generada en el proceso de firma del emisor, es la misma que la firma final del receptor, generada en el proceso de verificación, se garantiza la autenticidad del emisor, la integridad del mensaje y el no repudio en origen.

Aunque en la documentación del API, Java soporta también los algoritmos de curvas elípticas[2], no estaban incluidos en ninguno de los proveedores criptográficos (Crypto, BC y HarmonyJSSE) disponibles por defecto, por lo que no se pudo realizar el estudio de éstos.

[2] http://developer.android.com/reference/java/security/interfaces/package-summary.html

Para elaborar un análisis completo, se ha elaborado un estudio individualizado de cada algoritmo para finalmente realizar una comparativa entre ambos.

A continuación se muestra el orden que se seguirá para la presentación de resultados:

- Algoritmo RSA
    - Firma y verificación
    - Generación de claves
- Algoritmo DSA
    - Firma y verificación
    - Generación de claves
- Resumen y discusión

5.2. Algoritmo RSA

5.2.1. Firma y verificación

Evolución del tiempo de firma y verificación modificando la función hash

En todos los casos, independientemente de la función hash seleccionada, el tiempo de firma supera al tiempo de verificación. El factor diferencia irá variando en función de los parámetros seleccionados. Obsérvese como ejemplo la siguiente Figura que muestra el factor diferencia entre el tiempo de firma con respecto al de verificación del algoritmo RSA con MD4 de función hash. Se advierte como a medida que el tamaño de la clave aumenta, la diferencia entre estos tiempos asciende también, y esta diferencia se hace más notable a menor tamaño del fichero, de forma que el máximo que se alcanza es para el caso de un tamaño de clave de 2028 y un fichero de 1KB, resultando el tiempo de firma hasta 15 veces superior al tiempo de verificación. Por ello, el algoritmo RSA puede recomendarse en aplicaciones que realicen más operaciones de verificación que de firma.

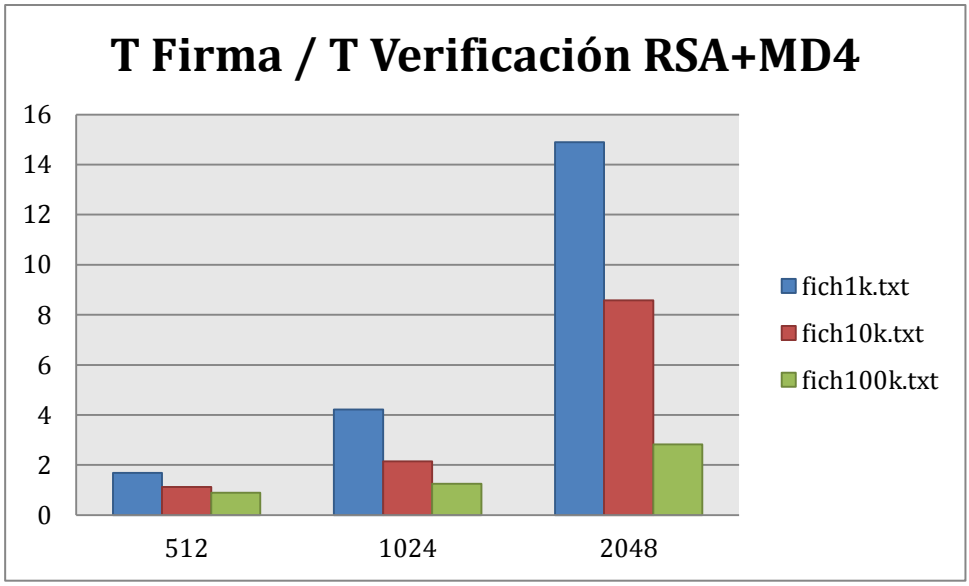


Figura 45. Factor diferencia tiempo firma con respecto tiempo verificación RSA con MD4

A la vista de los resultados, se aprecia como el tiempo de firma va ascendiendo a medida que el tamaño de la clave aumenta, en cambio el tiempo de verificación parece depender únicamente del tamaño del fichero, tomando valores similares independientemente de la longitud de la clave.

A continuación se destacan las particularidades de cada función hash:

MD4

Si el tamaño de la clave es pequeño, el tiempo de firma y verificación está muy próximo, tomando valores de magnitud similar, aunque la firma supera a la verificación. Según el tamaño de la clave va aumentando, esta diferencia se va haciendo más notoria. Fijando los valores obtenidos con una clave de 2048, para un fichero de 100KB, la firma es casi tres veces superior a la verificación, y este factor aumenta a 15 si el fichero es de 1KB.

Como ya se ha comentado en los aspectos generales, el tiempo de verificación parece estabilizarse con el tamaño del fichero, es decir, para un archivo de 100KB está sobre los 52ms, si el archivo es de 10KB 10.8 ms y si es de 1KB, 4.6 ms; en cambio, se produce un incremento del coste temporal frente a la subida en bits del tamaño de clave.

En cambio, el tiempo de firma depende del tamaño de la clave. Si el tamaño del fichero es 1KB, el tiempo en firmar con una clave de 1024 es el triple que si se usase un tamaño de 512, y seis veces mayor si la clave es de 2048. En ficheros de mayor tamaño, esta diferencia temporal no es tan acusada.

Tabla 17. Resultados RSA con MD4

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FACTOR
MD4	fich100k	512	0,047581667	0,053211166	0,89
MD4	fich10k	512	0,010607166	0,009406499	1,13

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FACTOR
MD4	fich1k	512	0,005093166	0,003030332	1,68
MD4	fich100k	1024	0,062940833	0,050132667	1,26
MD4	fich10k	1024	0,021909500	0,010213333	2,15
MD4	fich1k	1024	0,016449000	0,003902166	4,22
MD4	fich100k	2048	0,148300334	0,052634500	2,82
MD4	fich10k	2048	0,109500834	0,012773333	8,57
MD4	fich1k	2048	0,104324000	0,007005166	14,89

MD5

Si el tamaño de la clave es pequeño, el tiempo de firma y verificación está muy próximo, tomando valores de magnitud similar, aunque la firma supera a la verificación. Según el tamaño de la clave va aumentando, esta diferencia se va haciendo más notoria. Fijando la clave a 2048, para un fichero de 100KB, la firma es superior en un factor 2.5, y este factor aumenta a 14.5 si el fichero es de 1KB.

El tiempo de verificación para un archivo de 100KB está sobre los 59 ms, si el archivo es de 10KB 11.5 ms y si es de 1KB, 5 ms; no obstante, se produce un incremento del coste temporal frente a la subida en bits del tamaño de clave.

Con respecto a la evolución del tiempo de firma siendo el tamaño del fichero es 1KB, el tiempo en firmar con una clave de 1024 es mayor en un factor de 3.4 que si se usase un tamaño de 512, y en un factor de 5.7 si la clave es de 2048. En ficheros de mayor tamaño, esta diferencia temporal no es tan acusada.

Tabla 18. Resultados RSA con MD5

HASH	FICHERO	CLAVE	T.FIRMA	T.VERIFICACION	FACTOR
MD5	fich100k	512	0,05823083	0,05687900	1,02
MD5	fich10k	512	0,01246733	0,01026717	1,21
MD5	fich1k	512	0,00538400	0,00333033	1,62
MD5	fich100k	1024	0,06674750	0,05506900	1,21
MD5	fich10k	1024	0,02317183	0,01035783	2,24
MD5	fich1k	1024	0,01836817	0,00492167	3,73
MD5	fich100k	2048	0,16187533	0,06492517	2,49
MD5	fich10k	2048	0,11168783	0,01399883	7,98
MD5	fich1k	2048	0,10472417	0,00726900	14,41

SHA1

Para un tamaño de la clave de 512, el tiempo de firma y verificación está muy próximo, tomando valores de magnitud similar, aunque la firma supera a la verificación, siendo esta diferencia más notable a menor volumen de datos. Según el tamaño de la clave va aumentando, esta diferencia se va haciendo más notoria. Los máximos registrados se cumplen

fijando la clave a 2048. La firma de un fichero de 100KB es superior en un factor 2.5, y este factor aumenta a 14.2 si el fichero es de 1KB.

El tiempo de verificación es aproximadamente análogo con respecto al tamaño del fichero. Los resultados alcanzados para un archivo de 100KB están sobre los 57 ms, si el archivo es de 10KB 11.4 ms y si es de 1KB, 4.9 ms; denótese que estos son valores promedios, de forma que se produce un incremento del coste temporal frente a la subida en bits del tamaño de clave.

En el análisis temporal de la firma, siendo el tamaño del fichero es 1KB, el tiempo que se alcanza con el uso de una clave de 1024 es el triple que si se usase una clave de 512, y aumenta en un factor 6 si la clave es de 2048. Como se viene observando, en ficheros de mayor tamaño, esta diferencia temporal no es tan acusada.

Tabla 19. Resultados RSA con SHA1

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FACTOR
SHA1	fich100k	512	0,06027733	0,05378800	1,12
SHA1	fich10k	512	0,01196850	0,00987017	1,21
SHA1	fich1k	512	0,00543383	0,00324500	1,67
SHA1	fich100k	1024	0,07219767	0,05832333	1,24
SHA1	fich10k	1024	0,02319000	0,01045900	2,22
SHA1	fich1k	1024	0,01679700	0,00416450	4,03
SHA1	fich100k	2048	0,15597783	0,06105267	2,55
SHA1	fich10k	2048	0,11112517	0,01376633	8,07
SHA1	fich1k	2048	0,10421200	0,00731700	14,24

SHA224

El tiempo de firma y verificación está muy próximo para tamaños de clave 512 en un factor promedio de 1.2. Según aumenta el tamaño de la clave, esta diferencia se va haciendo más notoria, alcanzando máximos para un tamaño de clave de 2048 en un fichero de 100KB, de forma que la firma es superior en un factor 1.5, y este factor aumenta a 12.5 si el fichero es de 1KB.

El tiempo de verificación para un archivo de 100KB está sobre los 133 ms, si el archivo es de 10KB 21.8 ms y si es de 1KB, 6 ms.

El tiempo de firma con una clave de 1024 si el tamaño del fichero es 1KB es superior en un factor de 2.7 con respecto a usar una clave de 512, y en un factor de 6 si la clave es de 2048. En ficheros de mayor tamaño, esta diferencia temporal no es tan significativa.

Tabla 20. Resultados RSA con SHA224

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICACIÓN	FACTOR
SHA224	fich100k	512	0,13724083	0,13441850	1,02

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICACIÓN	FACTOR
SHA224	fich10k	512	0,02218983	0,02002367	1,11
SHA224	fich1k	512	0,00659950	0,00438433	1,51
SHA224	fich100k	1024	0,14556667	0,13018150	1,12
SHA224	fich10k	1024	0,03393750	0,02130867	1,59
SHA224	fich1k	1024	0,01797767	0,00520950	3,45
SHA224	fich100k	2048	0,23041167	0,13636433	1,69
SHA224	fich10k	2048	0,12110333	0,02409300	5,03
SHA224	fich1k	2048	0,10609267	0,00844967	12,56

SHA256

Si el tamaño de la clave es pequeño, el tiempo de firma y verificación es semejante aunque la firma supera a la verificación. Según el tamaño de la clave va aumentando, esta diferencia se va haciendo más evidente. Tomando los valores obtenidos fijando la clave a 2048, para un fichero de 100KB, la firma es superior en un factor 1.7, y este factor aumenta a 12.6 si el fichero es de 1KB.

El tiempo de verificación para un archivo de 100KB alcanza un promedio de 129.4 ms, si el archivo es de 10KB 21.4 ms y si es de 1KB, 6 ms.

Los valores máximos obtenidos entre la relación temporal del tiempo de firma se obtienen para un tamaño de datos de 1KB. En este caso, el tiempo de firma con una clave de 1024 es superior en un factor de 2.7 con respecto a usar una clave de 512, y en un factor de 6 si la clave es de 2048.

Tabla 21. Resultados RSA con SHA256

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FACTOR
SHA256	fich100k	512	0,13191433	0,13008283	1,01
SHA256	fich10k	512	0,02165583	0,01988133	1,09
SHA256	fich1k	512	0,00648450	0,00454750	1,43
SHA256	fich100k	1024	0,14239833	0,12722717	1,12
SHA256	fich10k	1024	0,03339567	0,02087567	1,60
SHA256	fich1k	1024	0,01778567	0,00516750	3,44
SHA256	fich100k	2048	0,22928883	0,13093583	1,75
SHA256	fich10k	2048	0,12109667	0,02354783	5,14
SHA256	fich1k	2048	0,10547050	0,00839250	12,57

SHA384

Con esta configuración, no es posible la utilización de una clave de 512b, por lo que sólo hay resultados para tamaños de 1024 y 2048. Si la clave es 1024, el tiempo de firma y verificación está muy próximo si el tamaño del archivo es de 100Mb, pero esta diferencia va aumentando según el volumen de datos se va reduciendo, de forma que para un archivo de 1KB el tiempo

de firma es el triple que el tiempo de verificación. Si se fija la clave a 2058, para un fichero de 100KB, la firma es superior en un factor 1.6, y este factor aumenta a 11.6 si el fichero es de 1KB.

Como ya se ha comentado en los aspectos generales, el tiempo de verificación parece depender principalmente del tamaño del fichero, por lo que si el archivo es de 100KB, la verificación alcanza 169 ms, si el archivo es de 10KB 27.4 ms y si es de 1KB, 7.6 ms.

El tiempo de firma varía en función del fichero y clave, por lo que si el tamaño del fichero es 1KB, con una clave de 2048 es superior en un factor de 5.6 con respecto a usar una clave de 1024. En ficheros de mayor tamaño, esta diferencia temporal no es tan acusada.

Tabla 22. Resultados RSA con SHA384

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FACTOR
SHA384	fich100k.txt	1024	0,18098483	0,16796300	1,08
SHA384	fich10k.txt	1024	0,03840433	0,02603250	1,48
SHA384	fich1k.txt	1024	0,01894133	0,00602433	3,14
SHA384	fich100k.txt	2048	0,26758167	0,17062183	1,57
SHA384	fich10k.txt	2048	0,13058283	0,02877983	4,54
SHA384	fich1k.txt	2048	0,10666483	0,00915583	11,65

SHA512

Con esta configuración, no es posible la utilización de una clave de 512b, por lo que sólo hay resultados para tamaños de 1024 y 2048. Si la clave es 1024, el tiempo de firma y verificación es muy cercano para tamaños de archivo grandes, pero esta diferencia va aumentando según el volumen de datos se va reduciendo y aumenta el tamaño de la clave, de forma que para un archivo de 1KB el tiempo de firma es el triple que el tiempo de verificación, y este factor aumenta a 11.5 para una clave de 2048.

El tiempo de verificación en un archivo de 100KB asciende a los 172 ms, si el archivo es de 10KB 27.5 ms y si es de 1KB, 7.8 ms.

Observando la firma, para un tamaño de fichero de 1KB y con una clave de 2048, el tiempo es superior en un factor de 5.6 con respecto a usar una clave de 1024. En ficheros de mayor tamaño, esta diferencia temporal no es tan acusada.

Tabla 23. Resultados RSA con SHA512

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FIRMA
SHA512	fich100k.txt	1024	0,18170067	0,17368117	1,05
SHA512	fich10k.txt	1024	0,03857950	0,02605450	1,48
SHA512	fich1k.txt	1024	0,01888767	0,00644833	2,93
SHA512	fich100k.txt	2048	0,26768067	0,17072650	1,57
SHA512	fich10k.txt	2048	0,12620783	0,02889567	4,37

HASH	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FIRMA
SHA512	fich1k.txt	2048	0,10672267	0,00923833	11,55

Evolución del tiempo de Firma y Verificación en función del tamaño del fichero

Como se viene observando, el tiempo de firma siempre supera al de verificación, y a mayor complejidad de algoritmo de firma, mayor es el tiempo de ambos.

A la vista de los resultados, se puede realizar una agrupación de las funciones hash en función del rango temporal obtenido. De esta forma, el comportamiento de MD4, MD5 y SHA1 está muy próximo, en un segundo grupo se asociaría SHA224 y SHA256, y los valores más altos alcanzados se han alcanzado en una tercera categoría, en la que se encuentran SHA284 y SHA512.

A continuación se detalla el análisis en función del tamaño del archivo elegido.

Grande: 100KB

A la vista de las figuras 2 y 3, los resultados se pueden agrupar en tres grupos:

- Grupo 1: MD4, MD5 y SHA1, con valores de firma y verificación para una clave de 2048 están estimados en 155 ms y 60 ms, si la clave es de 1024, 67 ms y 54.5 ms y si la clave es de 512 en torno a 55 ms y 54.6 ms respectivamente.

- Grupo 2: SHA224 y SHA256, con valores de firma y verificación de 230 ms y 133.6 ms si la clave es 2048, 144 ms y 129 ms si la clave es de 1024, y 134 ms y 132 si la clave es de 512.

- Grupo 3: SHA384 y SHA512, con valores de firma y verificación en torno a 267 ms y 170 ms si la clave es de 2048 o 181 ms y 170 ms si la clave es de 1024.

A partir de estos resultados, se observa como en cada uno de los grupos, el tiempo de verificación es muy próximo independientemente del tamaño de la clave utilizada, por lo que la diferencia entre usar uno u otro algoritmo lo marca principalmente el tiempo de firma que sí es dependiente de la clave.

Se observa como el tiempo de firma y verificación puede triplicarse entre utilizar alguna función hash del primer grupo con respecto al tercero.

Mediano: 10KB

A la vista de las figuras 4 y 5, se pueden agrupar los resultados en tres grupos en función de los resultados temporales:

- Grupo 1: MD4, MD5 y SHA1, con valores de firma y verificación para una clave de 2048 están estimados en 111 ms y 13.5 ms, si la clave es de 1024, 22.7 ms y 10.3 ms y si la clave es de 512 en torno a 11.7 ms y 9.8 ms respectivamente.

- Grupo 2: SHA224 y SHA256, con valores de firma y verificación de 121 ms y 24 ms si la clave es 2048, 33.6 ms y 21 ms si la clave es de 1024, y 22 ms y 20 si la clave es de 512.

- Grupo 3: SHA384 y SHA512, con valores de firma y verificación en torno a 128 ms y 28.8 ms si la clave es de 2048 o 38.5 ms y 26 ms si la clave es de 1024.

A partir de estos resultados, se observa como en cada uno de los grupos, el tiempo de verificación es muy cercano independientemente del tamaño de la clave utilizada, por lo que la diferencia entre usar uno u otro algoritmo lo marca principalmente el tiempo de firma.

Se observa como el tiempo de firma no llega a ser el doble entre usar una función resumen del primer grupo con respecto del tercero. Por lo general los tiempos de firma, manteniendo fijo el tamaño de la clave, no están tan distanciados. Esta diferencia si se puede apreciar en mayor medida en el tiempo de verificación, de forma que la diferencia temporal entre elegir cualquier algoritmo del grupo 1 puede ser el doble con respecto a la elección de un algoritmo del grupo 3, bajo las mismas circunstancias.

Pequeño: 1KB

A partir de las figuras 6 y 7, se pueden agrupar los resultados en tres grupos:

- Grupo 1: MD4, MD5 y SHA1, con valores de firma y verificación para una clave de 2048 están estimados en 104 ms y 7 ms, si la clave es de 1024, 17 ms y 4.5 ms y si la clave es de 512 en torno a 5 ms y 3 ms respectivamente.

- Grupo 2: SHA224 y SHA256, con valores de firma y verificación de 105.8 ms y 8.3 ms si la clave es 2048, 17.8 ms y 5 ms si la clave es de 1024, y 6.5 ms y 4.4 si la clave es de 512.

- Grupo 3: SHA384 y SHA512, con valores de firma y verificación en torno a 106 ms y 9 ms si la clave es de 2048 o 18.9 ms y 6.2 ms si la clave es de 1024.

A partir de estos resultados, se observa que tanto el tiempo de firma y el de verificación son muy cercanos entre los tres grupos, aunque al tomar valores tan pequeños (inferiores a 10 ms) la diferencia no es tan perceptible. No obstante, la diferencia temporal depende directamente del tamaño de la clave. El tiempo de firma si la clave es de 2048 es superior en un factor de aproximadamente 5.8 con respecto al uso de una clave de 1024.

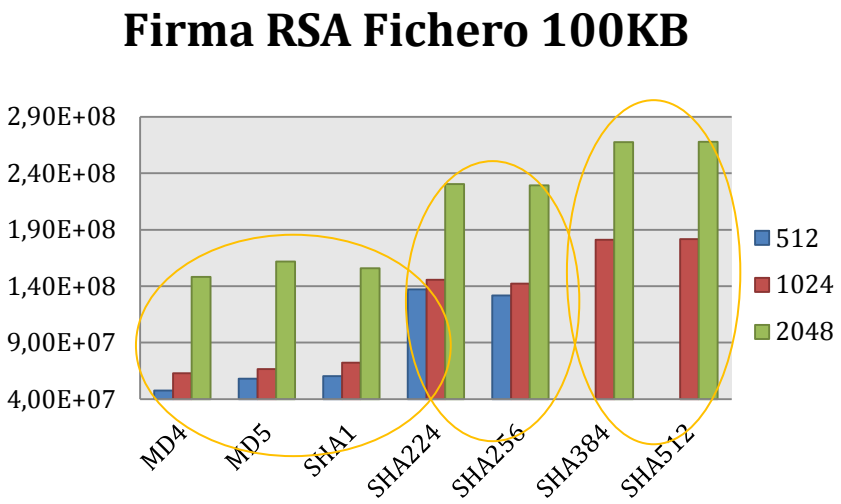


Figura 46. Resultados firma RSA fichero 100KB

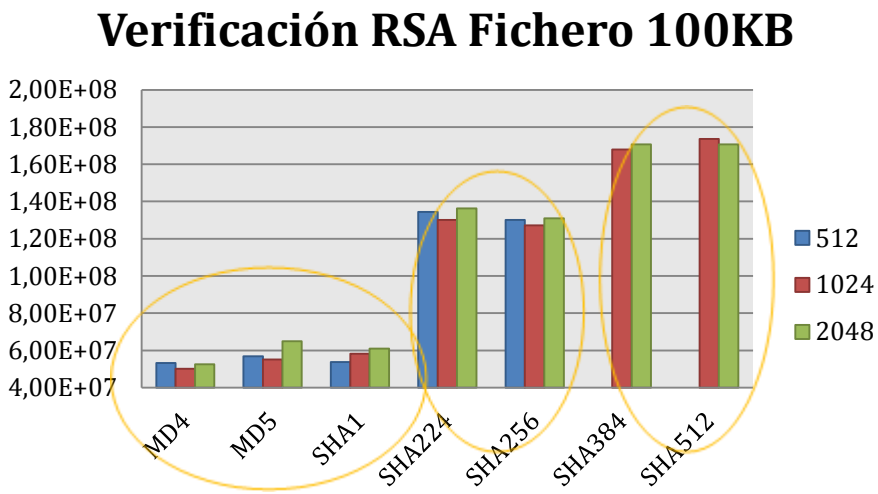


Figura 47. Resultados verificación RSA fichero 100KB

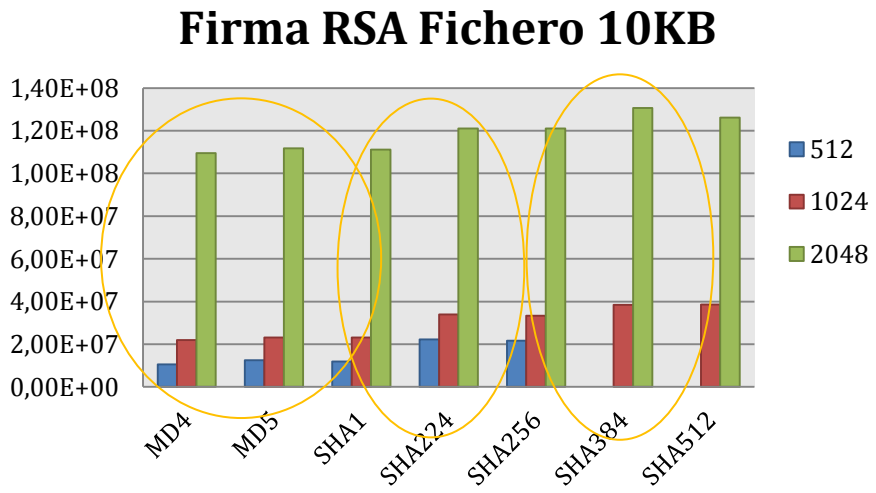


Figura 48 Resultados firma RSA fichero 10KB

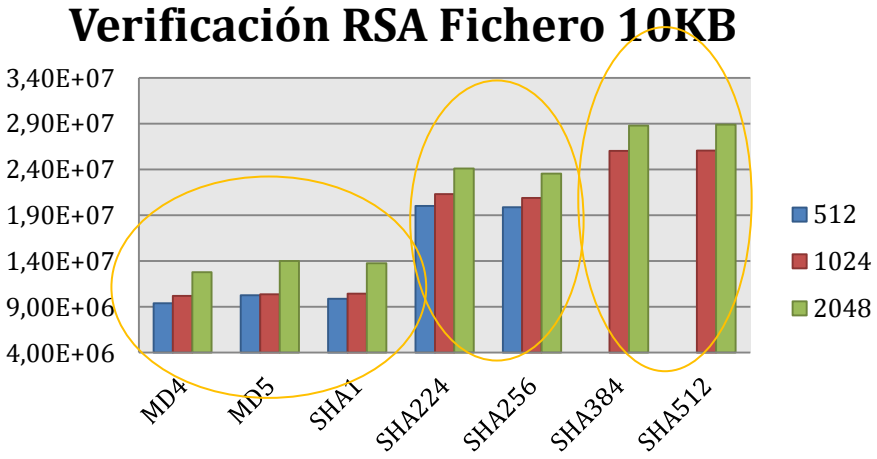


Figura 49. Resultados verificación RSA fichero 10KB

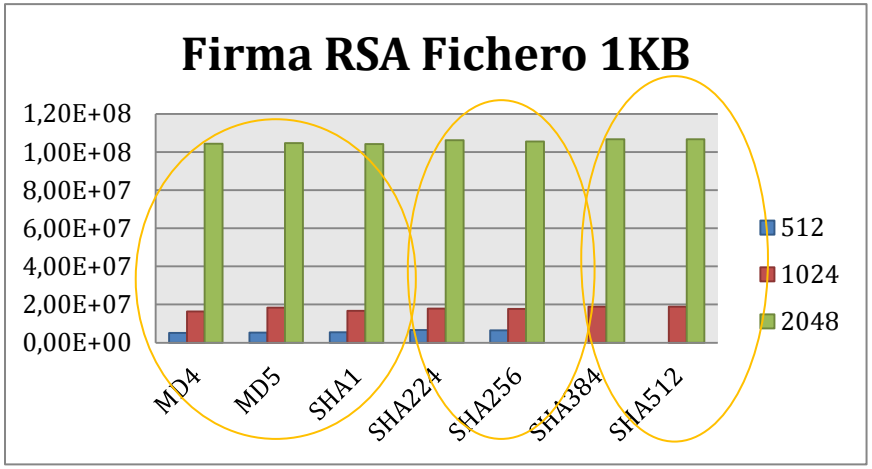


Figura 50. Resultados firma RSA fichero 1KB

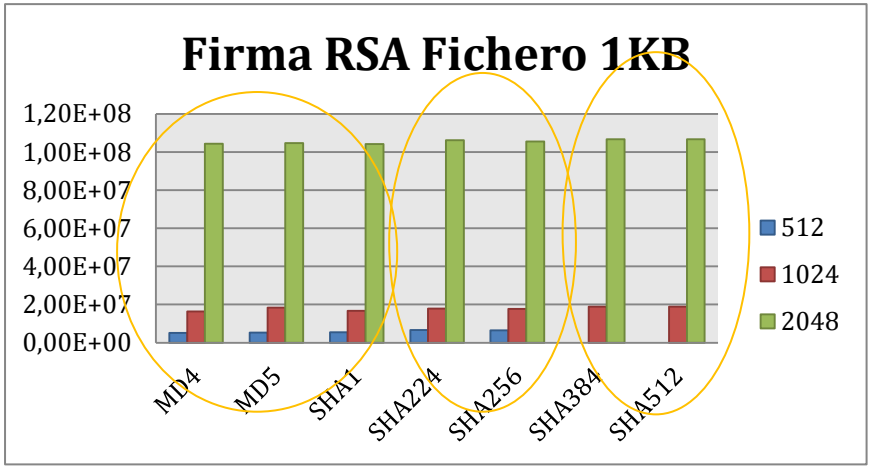


Figura 51. Resultados verificación RSA fichero 1KB

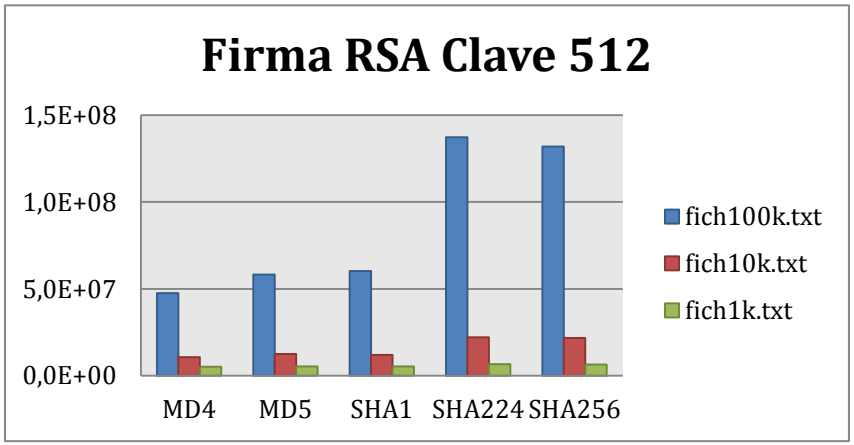


Figura 52. Resultados firma RSA clave 512

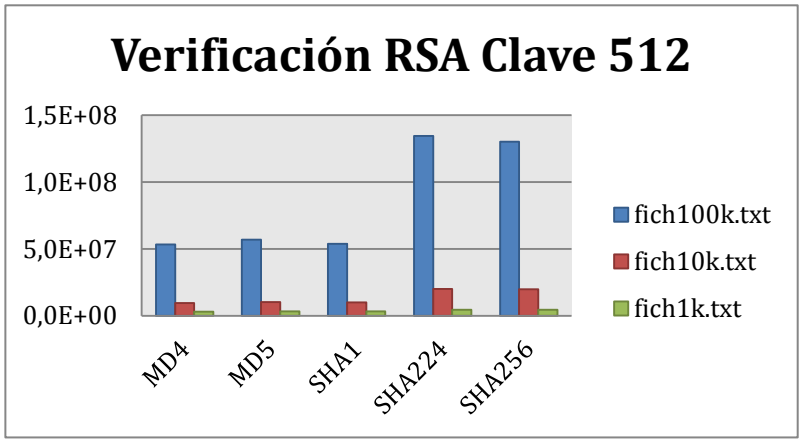


Figura 53. Resultados verificación RSA clave 512

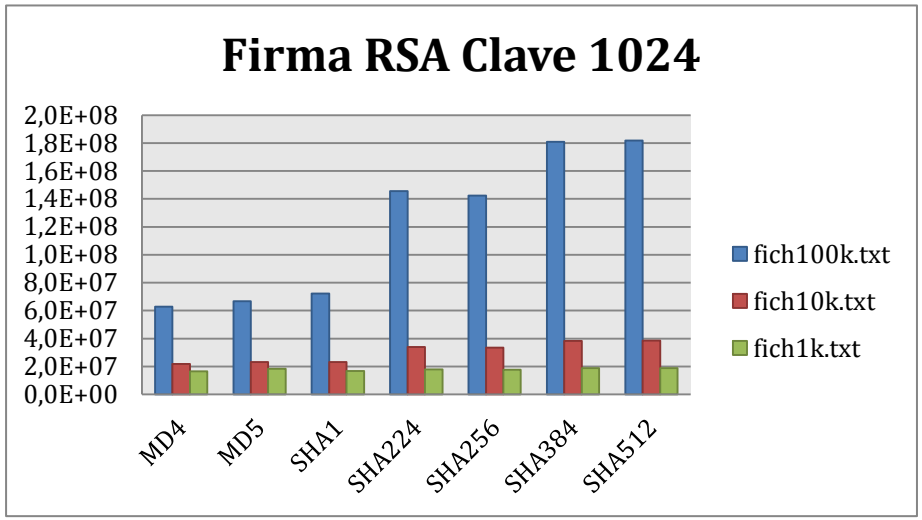


Figura 54. Resultados firma RSA clave 1024

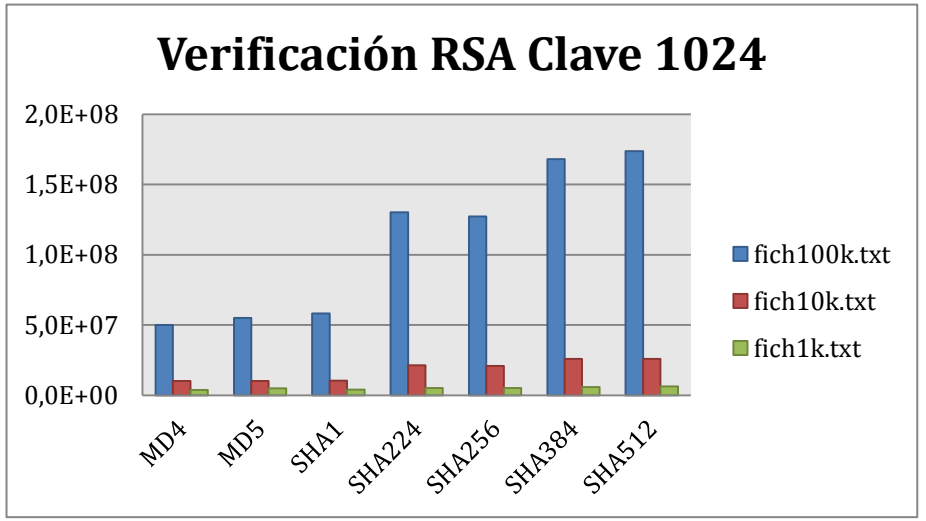


Figura 55. Resultados verificación RSA clave 1024

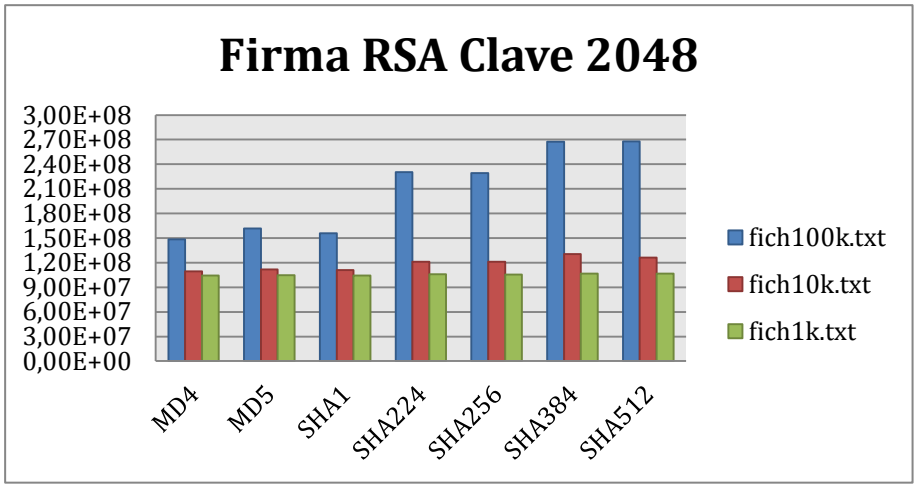


Figura 56. Resultados firma RSA clave 2048

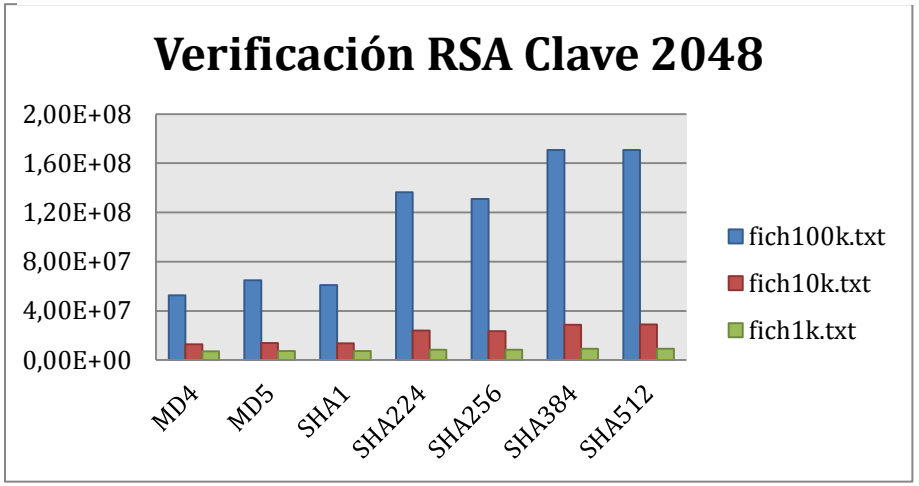


Figura 57. Resultados verificación RSA clave 2048

Evolución del tiempo de Firma y Verificación en función del tamaño de la clave

Clave 512

Se observa como los tiempos de firma y verificación están en una relación 1:1, aunque ligeramente superiores en el caso de firma (véase figuras 52 y 53).

Clave 1024

Si el tamaño del fichero es grande, los tiempos de firma y verificación son muy próximos, aunque ligeramente inferiores en el caso de verificación. No obstante, para tamaños de datos inferiores, comienza a observarse que el tiempo de firma es el doble del tiempo de verificación (véase figuras 54 y 55).

Clave 2048

Para un tamaño de clave grande, se hace muy notable la diferencia entre el tiempo de firma y verificación, encontrándose los menores factores de relación en un fichero grande. Por el contrario, cuanto menor es el tamaño de los datos a procesar, el factor diferencia es el mayor, entre 12 y 15 veces (véase figuras 56 y 57).

5.2.2. Generación del par de claves

En cualquier caso, el tiempo de generación de claves solo dependerá del tamaño de la clave seleccionada, siendo independiente de la función hash o incluso del tamaño del fichero.

Análisis tiempo generación par de claves RSA

El tiempo en generar el par de claves de tamaño 1024 (1.51 seg) es superior en un factor de 4.5 con respecto a una clave de la mitad del tamaño (334 ms). Si la comparación se realiza entre una clave de 2048 con respecto a 1024, este factor aumenta a 5.6 ya que se necesitan una media de 8.5 segundos en poder ser generadas.

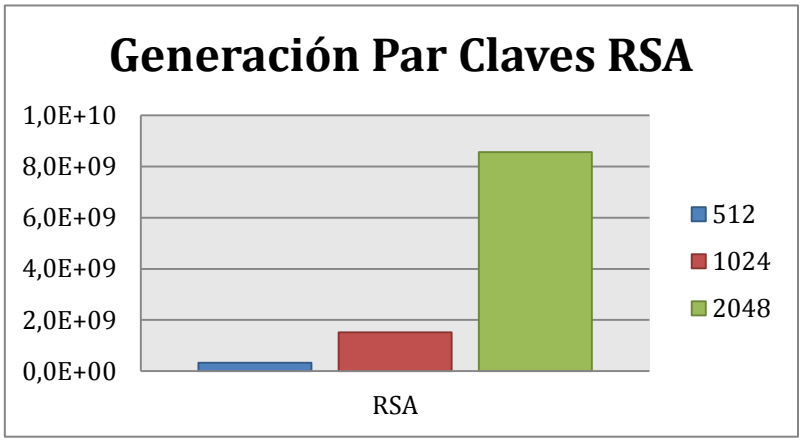


Figura 58. Tiempo generación claves RSA

5.3. Algoritmo DSA

5.3.1. Firma y verificación

Evolución del tiempo de Firma y Verificación del algoritmo DSA en función del hash

En todos los casos, independientemente de la función hash seleccionada, el tiempo de verificación supera al tiempo de firma, al contrario que RSA. El factor de diferencia irá variando en función de los parámetros seleccionados, pero es muy inferior ya que varía únicamente entre 1 y 1.8 aproximadamente. Por ello, el algoritmo DSA puede recomendarse en aplicaciones que realicen más operaciones de firma que de verificación.

A continuación se destacan las particularidades de cada función hash:

SHA1

Solo se han evaluado dos tamaños de clave: 512 y 1024.

Para volúmenes de datos grandes el tiempo de firma y verificación está muy próximo, tomando valores de magnitud similar, aunque la firma es inferior a la verificación. No obstante, según el tamaño de la clave va aumentando, esta diferencia se va incrementando ligeramente. Tomando los valores obtenidos fijando la clave a 1024, para un fichero de 100KB, la verificación es superior en un factor 1.35, y este factor aumenta a 1.76 si el fichero es de 1KB.

None

Si no se aplica función hash, los tiempos obtenidos son superiores dado que no se está realizando el resumen. Para volúmenes de datos grandes el tiempo de firma y verificación está muy próximo, tomando valores de magnitud similar, aunque la firma es inferior a la verificación. No obstante, según el tamaño de la clave va aumentando, esta diferencia se va incrementando ligeramente. Tomando los valores obtenidos fijando la clave a 1024, para un fichero de 100KB, la verificación es superior en un factor 1.02, y este factor aumenta a 1.75 si el fichero es de 1KB.

Tabla 24. Resultados firma y verificación RSA

MODO	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FACTOR
SHA1	fich100k.txt	512	0,03031883	0,03253300	1,07
SHA1	fich10k.txt	512	0,00809400	0,01020117	1,26
SHA1	fich1k.txt	512	0,00483517	0,00695550	1,44
SHA1	fich100k.txt	1024	0,03624500	0,04898317	1,35
SHA1	fich10k.txt	1024	0,01404700	0,02217117	1,58
SHA1	fich1k.txt	1024	0,01084350	0,01905550	0,43
None	fich100k.txt	512	0,12890817	0,12216033	4,25
None	fich10k.txt	512	0,01166933	0,01361017	1,17
None	fich1k.txt	512	0,00520200	0,00712450	1,37

MODO	FICHERO	CLAVE	T.FIRMAR	T.VERIFICAR	FACTOR
None	fich100k.txt	1024	0,13071900	0,13320483	3,61
None	fich10k.txt	1024	0,01802283	0,02532650	1,41
None	fich1k.txt	1024	0,01123067	0,01971300	0,43

Evolución del tiempo de Firma y Verificación en función del tamaño del fichero

Como se observa de los gráficos, y de las observaciones ya realizadas tras su estudio individualizado, el tiempo de verificación supera al de firma.

Para volúmenes grandes de datos, no parece existir una gran diferencia entre aplicar SHA1 y None, aunque el uso de una la función resumen siempre produce resultados inferiores. Según el fichero de datos se va reduciendo, esta diferencia sí se hace más notoria.

A continuación se detalla el análisis en función del tamaño del archivo elegido.

Grande: 100KB

Para un mismo tamaño de clave, por ejemplo 512, el tiempo de firma y verificación puede cuadriplicarse en caso de utilizar la función hash o no, aunque esta diferencia se reduce un poco si el tamaño de la clave es mayor, en un factor generalizado de 3.2.

Si se firma con la función SHA1, la diferencia entre la firma y la verificación es de unos 2.21 ms si se utiliza una clave de 512, mientras que si la clave es de 1024, esta diferencia aumenta hasta casi los 13 ms. Se observa que el porcentaje que supone esta diferencia temporal respecto al tiempo de verificación pasa de un 6.8% a un 26%.

Si no se utiliza una función resumen, la diferencia entre la firma y verificación es de unos 6.75 ms si se utiliza una clave de 512 y 2.49 ms si la clave es 1024. Este tiempo solo supone un 5.5% y un 1.9% con respecto al tiempo de verificación.

Mediano 10KB

La diferencia entre la firma y la verificación no es tan grande, sino que varía entre un factor de 1.1 o 1.6. La mayor diferencia se encuentra usando SHA1 con un tamaño de clave de 1024, que asciende hasta 8 ms, lo que supone un 36.6% respecto al tiempo de verificación. Si no se utiliza hash, esta diferencia temporal es de 7.3 ms, un 28.8% con respecto al tiempo de verificación. Si la clave es de 512, esta diferencia está en torno a los 2 ms se utilice hash o no, aunque en todas las casuísticas se obtienen resultados más altos si no se utiliza función resumen.

Si se firma con la función SHA1, la diferencia entre la firma y la verificación es de unos 2.11 ms si se utiliza una clave de 512, mientras que si la clave es de 1024, esta diferencia aumenta hasta casi los 8.12 ms. Se observa que el porcentaje que supone esta diferencia temporal respecto al tiempo de verificación pasa de un 20.6% a un 36.6%.

Si no se utiliza una función resumen, la diferencia entre la firma y verificación es de unos 1.94 ms si se utiliza una clave de 512 y 7.3 ms si la clave es 1024. Este tiempo supone un 14.3% y un 28.8% con respecto al tiempo de verificación.

El tiempo de firma aumenta en un factor 1.5 si se elige no utilizar hash frente a SHA1.

Pequeño 1KB

La diferencia entre la firma y la verificación varía en un factor de 1.4 y 1.75. La mayor diferencia se encuentra usando un tamaño de clave de 1024 para cualquiera de las dos opciones, superando los 8.2 ms, lo que supone un 43% respecto al tiempo de verificación. Si la clave es de 512, esta diferencia está en torno a los 2 ms se utilice hash o no, aunque en todas las casuísticas se obtienen resultados más altos si no se utiliza función resumen.

Si se firma con la función SHA1, la diferencia entre la firma y la verificación es de unos 2.12 ms si se utiliza una clave de 512, mientras que si la clave es de 1024, esta diferencia aumenta hasta casi los 8.21 ms. Se observa que el porcentaje que supone esta diferencia temporal respecto al tiempo de verificación pasa de un 30.5% a un 43.1%.

Si no se utiliza una función resumen, la diferencia entre la firma y verificación es de unos 1.37 ms si se utiliza una clave de 512 y 1.75 ms si la clave es 1024. Este tiempo supone entre un 27% y un 43% con respecto al tiempo de verificación.

El tiempo de firma puede llegar a duplicarse si se elige no utilizar hash frente a SHA1.

Evolución del tiempo de Firma y Verificación en función del tamaño de la clave

En ficheros grandes, los tiempos de firma y verificación obtenidos son muy parecidos, independientemente del tamaño de la clave, diferenciándose en unos 2ms, aunque siempre ligeramente superiores a mayor clave.

Para volúmenes de datos inferiores a 10KB, la diferencia entre usar una clave de 512 y 1024 se hace más notoria.

Clave 512

Firmar con SHA1 y no utilizar función resumen en archivos de gran tamaño supone que el coste temporal se cuadriplique para los dos tamaños de clave.

La diferencia temporal entre firmar un fichero de 10KB con respecto a uno de 1KB es de aproximadamente 1.57 si se utiliza SHA1 y asciende hasta 2 si no se utiliza función resumen.

La diferencia entre la firma y la verificación está en unos 2 ms para todas las opciones posibles.

Clave 1024

La diferencia entre usar hash o no para ficheros grande influye en gran medida en los resultados, de forma que los tiempos obtenidos sin usar resumen son unas tres veces superiores a su uso con hash, esto se traduce en términos temporales en unos 90 ms de diferencia.

Si el archivo es inferior a 10Kb, la diferencia entre usar hash o no, no es tan acusada: para un archivo mediano está en torno a los 3.5 ms, pero si el archivo es de 1KB, es inferior a 1 ms.

La diferencia temporal entre firmar un fichero de 10Kb con respecto a otro de 1KB es de unos 3 ms con SHA1, y de unos 6 ms si no se utiliza función hash.

La diferencia entre la firma y la verificación está en unos 8 ms tanto para archivos de 1Kb como de 10Kb, y con o sin función hash.

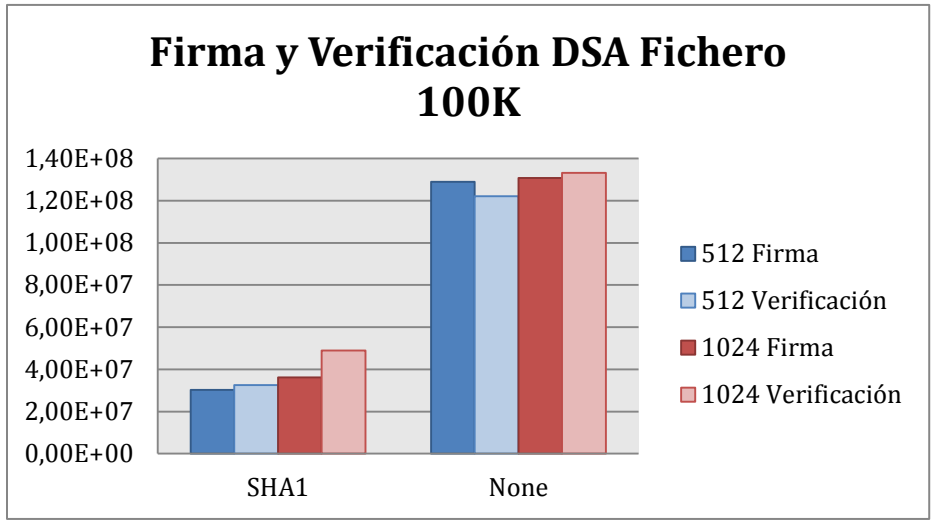


Figura 59. Resultados firma y verificación DSA fichero 100KB

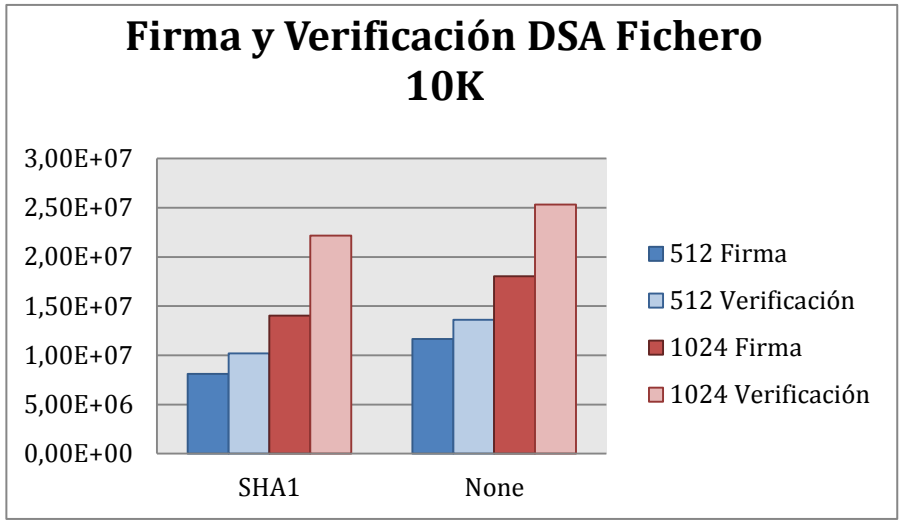


Figura 60. Resultados firma y verificación DSA fichero 10KB

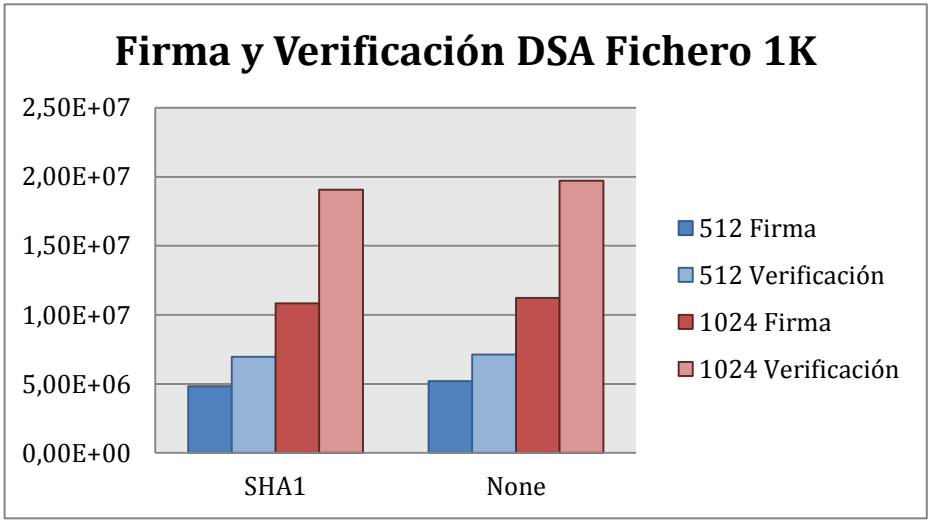


Figura 61. Resultados firma y verificación DSA fichero 1KB

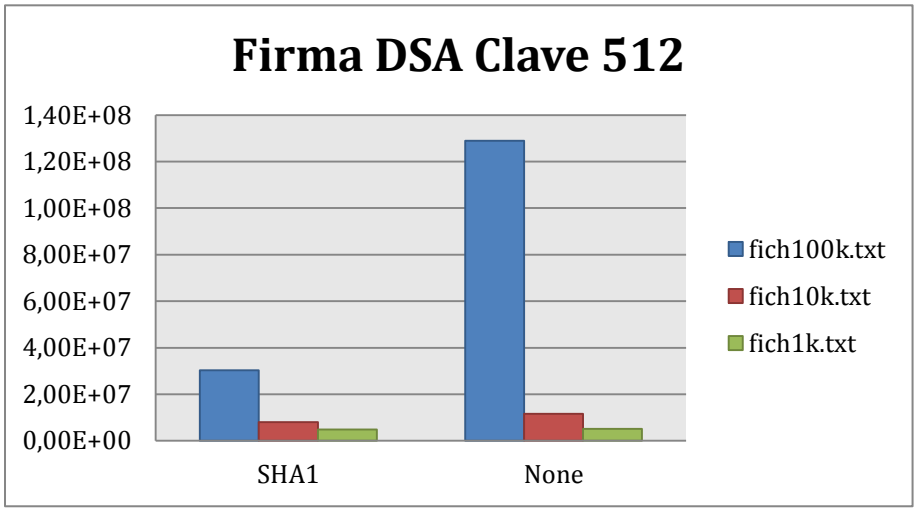


Figura 62. Resultados firma DSA clave 512

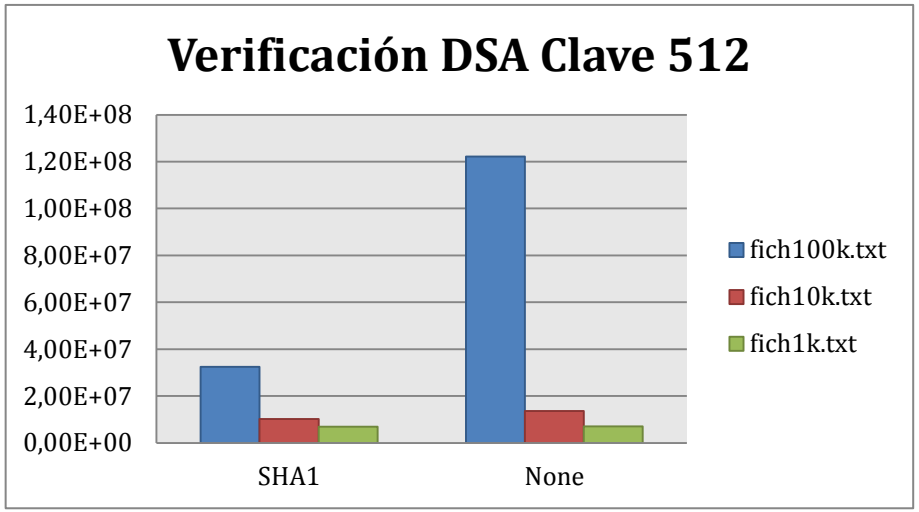


Figura 63. Resultados verificación DSA clave 512

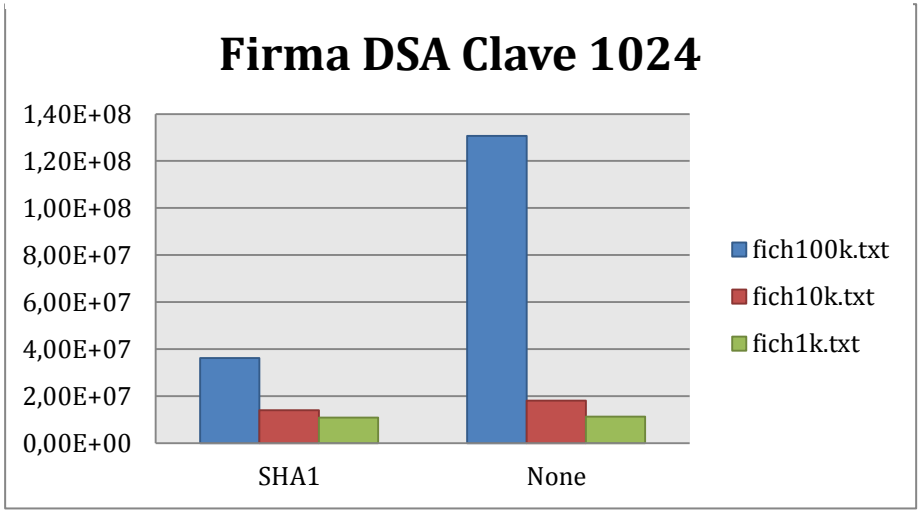


Figura 64. Resultados firma DSA clave 1024

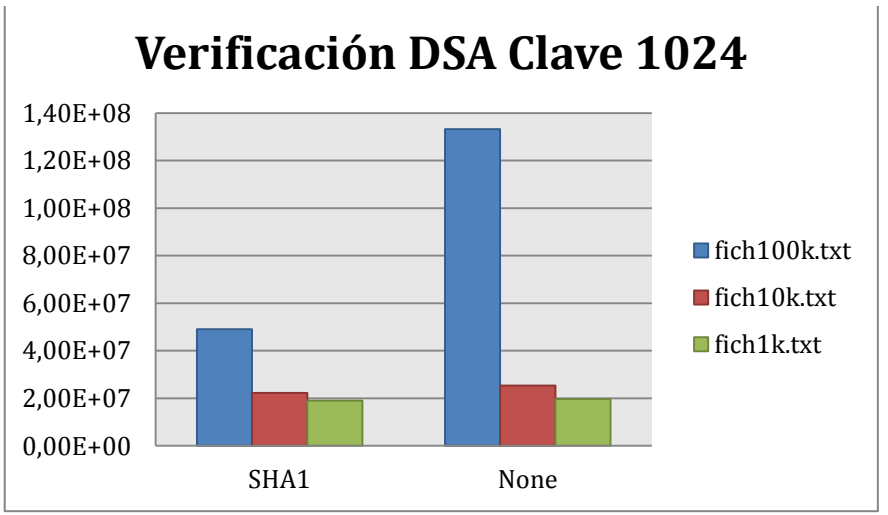


Figura 65. Resultados verificación DSA clave 1024

5.3.2. Generación del par de claves

En cualquier caso, el tiempo de generación de claves solo dependerá del tamaño de la clave seleccionada, siendo independiente de la función hash o incluso del tamaño del fichero.

Análisis tiempo de generación par de claves DSA

El tiempo en generar un par de claves de 1024 es hasta ocho veces más que si el tamaño de la clave fuese 512; en términos temporales esto se traduce en que son necesarios 934 ms para generar una clave de 512 y más de 7.76 segundos si la clave es de 1024.

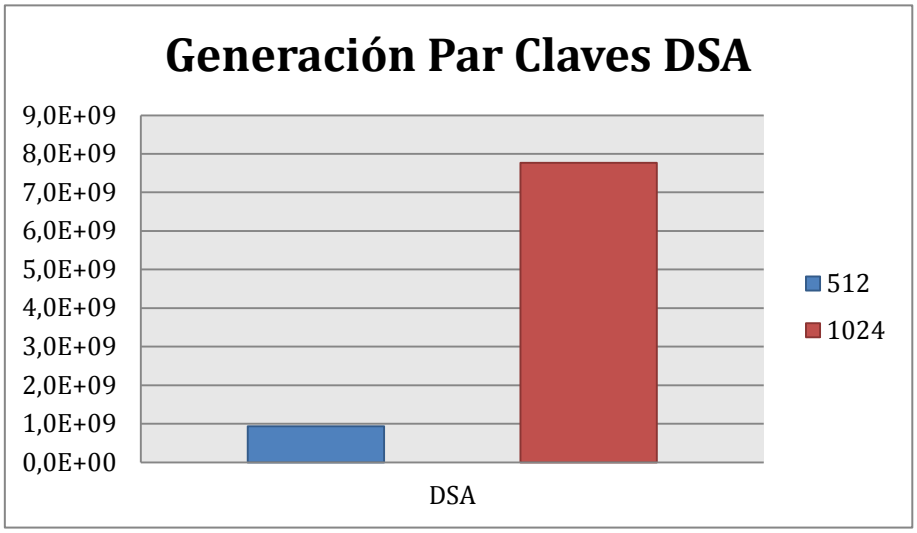


Figura 66. Tiempo generación par claves DSA

5.4. Resumen y discusión

Firma y Verificación. RSA variando función hash

Conclusiones

Si el tamaño de la clave es de 512, longitud pequeña, la relación del tiempo de firma con respecto al de verificación es de generalmente 1:1, por lo que no hay una gran diferencia temporal entre ambos resultados. No obstante, según va aumentando el tamaño de la clave, esta diferencia se va haciendo más notable, obteniéndose los mayores coeficientes cuanto menor es el volumen de datos a tratar, alcanzando factores máximos de hasta 14,8:1. Bajo esta observación, las funciones resumen MD4, MD5 y SHA1 tienen comportamientos semejantes, mientras que a partir de SHA224 este coeficiente va disminuyendo paulatinamente.

Si la observación se centra en los valores temporales obtenidos en la firma, se verifica como para datos de 1KB de longitud, el comportamiento de las distintas funciones hash es lineal, no apreciándose diferencias significativas entre usar una u otra. Esta diferenciación se advierte según aumenta el volumen de datos, apreciándose el aumento temporal a partir del uso de la función SHA224 y superiores, con respecto MD4, MD5 y SHA1, que siguen manteniendo

unos resultados semejantes. Las diferencias más significativas acontecen para un volumen de datos grande, considerándose tres escalones temporales en los que se pueden agrupar las funciones hash: un primer escalón con los resultados de MD4, MD5 y SHA1, un segundo escalón con SHA224 y SHA256, y el último con SHA384 y SHA512. El coste temporal que se debe asumir entre usar un escalón u otro puede alcanzar un factor 2.9 para una clave de 1024 (tiempo registrado con SHA512 es prácticamente tres veces superior a SHA1). Los resultados más rápidos son los que corresponden a las funciones hash del primer escalón.

Si el análisis se centra en los tiempos de verificación, como sucedía con la firma, para tamaños de datos pequeños, 1KB, la respuesta de las diferentes funciones hash es muy lineal. Las diferencias entre el uso de una u otra función resumen comienza a apreciarse con el aumento de volumen de datos, observándose el comportamiento escalonado en archivos de 100KB. El coste temporal que se debe asumir de usar SHA512 frente a SHA1 es superior a 3.2. Al igual que en el tiempo de firma, los resultados más rápidos que se han registrado son los correspondientes a las funciones MD4, MD5 y SHA1.

Generación del par de claves

Con un tamaño de la clave de 512, el algoritmo DSA es 2.79 veces más lento que el RSA, y esto se traduce en una diferencia temporal de unos 600 ms, siendo 334 ms el tiempo en generarse con RSA.

Si el tamaño de la clave es 1024, el algoritmo DSA es más de 5 veces más lento que el RSA, y esto se traduce en una diferencia temporal de más de 6 segundos, siendo 1.5 segundos el tiempo que es necesario para que se genere en con RSA.

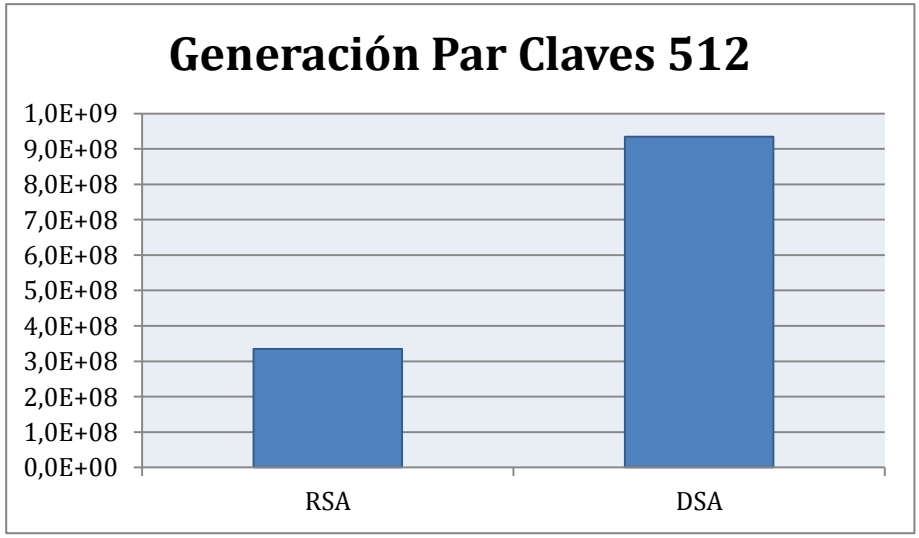


Figura 67. Tiempo generación par de claves 512

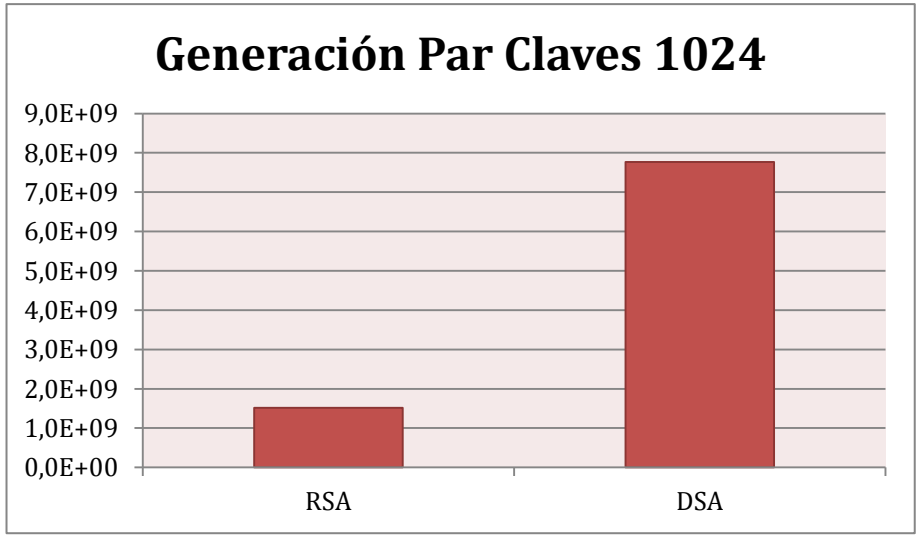


Figura 68. Tiempo generación par claves 1024

Capítulo 6

Estudio de la eficiencia de conexiones SSL

Una conexión segura se sirve del cifrado de datos y claves para garantizar la protección de los datos transmitidos entre un origen y un destino. Este capítulo tiene como propósito presentar el estudio del soporte y eficiencia que ofrece Android para el establecimiento de conexiones seguras entre cliente y servidor, a través del protocolo SSL.

.

6.1. Objetivos y metodología

El objetivo de este capítulo es realizar un estudio tanto del soporte como la eficiencia de los algoritmos de cifrado que aceptan los diferentes navegadores instalables de Android para el establecimiento de conexiones seguras mediante el protocolo de seguridad SSL.

Este estudio se ha realizado definiendo cuatro tareas:

1. Estudio del soporte de algoritmos de cifrado tanto en servidor, proporcionado por OpenSSL, como en cliente, teniendo en cuenta tanto el navegador nativo del dispositivo móvil como por otros navegadores instalados (LISTA NAVEGADORES). Posteriormente se realiza una comparativa del soporte ofrecido a partir de los resultados obtenidos.
2. Estudio del tiempo de descarga de ficheros de diferentes tamaños (1KB, 10 KB, 100KB y 1MB) de los cifradores soportados en cliente y servidor.
3. Estudio de la eficiencia de las conexiones, medida a partir de la sobrecarga introducida frente a la transmisión de los datos en claro para cada uno de los diferentes algoritmos de cifrado y navegador.
4. Estudio comparativo a partir de los resultados obtenidos del navegador nativo y el resto de navegadores instalados en el dispositivo móvil.

Para la realización de este estudio ha sido necesario preparar el siguiente escenario de pruebas que se describe a continuación, con el objetivo de crear una red aislada servidor web – router – dispositivo móvil.

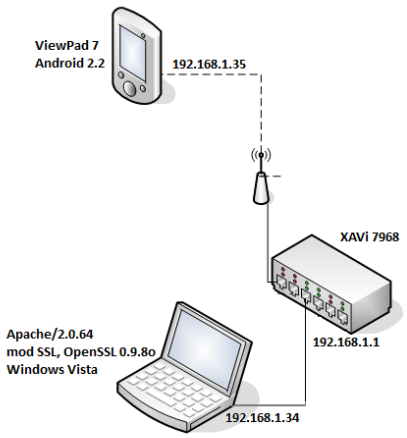


Figura 69. Escenario de pruebas

Los componentes de la red son: un servidor web Apache/2.0.64 (Win32) con el módulo SSL de OpenSSL 0.9.8o, instalado en un portátil con sistema operativo Windows Vista™ Home Premium; un router inalámbrico modelo XAVi 7968, responsable de asignar las direcciones IP a todos los componentes de la red, que permanece conectado al portátil a través de un cable Ethernet categoría 5; El último elemento de esta red lo

conforma el cliente que solicita las peticiones al servidor, y será un dispositivo móvil, modelo ViewPad 7 de ViewSonic con Android 2.2, conectado por WiFi a la red.

Para crear una red totalmente aislada se configura el router inalámbrico de forma que únicamente admita la MAC del dispositivo móvil. Esta restricción se realiza a través de la interfaz web del router (IP: 192.168.1.1).

La ruta para consultar la MAC del dispositivo móvil se encuentra en Ajustes / Acerca del teléfono / Estado / Dirección MAC de WiFi.

6.2. Soporte de cifradores

El servidor Apache instalado en el ordenador es el encargado de fijar el tipo de cifrado que se va a utilizar en cada conexión. Para cada conexión, se ha de modificar manualmente la directiva `SSLCipherSuite` en el fichero de configuración `httpd.conf` del servidor Apache. Esta directiva establece los protocolos de autenticación y cifrado que admite el servidor web, y está compuesta por cuatro atributos principales:

- Intercambio de claves: RSA o DH
- Algoritmo de autenticación: RSA, DH, DSS o ninguno
- Algoritmo de cifrado: DES, Triple-DES, RC4, RC2, IDEA o ninguno
- MAC Digest: MD5 o SHA1.

Las medidas han consistido en el establecimiento de una conexión segura a través del protocolo HTTPS forzando en la configuración del Apache a un cifrador específico y así poder comprobar la compatibilidad de algoritmos en el cliente.

6.2.1. Estudio cifradores soportados en OpenSSL

El primer hito consiste en conocer qué algoritmos de cifrado están disponibles en el servidor. Para ello, desde la consola de comandos, se accede a la ruta donde se encuentra instalado el servidor, en este caso en `c:\apache\apache2\bin` y se ejecuta la siguiente instrucción:

Openssl ciphers

Se muestra en la Tabla 25 el listado resultante de algoritmos de cifrado soportados por en el servidor, proporcionados a través de OpenSSL v 0.9.8:

Tabla 25. Listado cifradores soportados en OpenSSL

CIPHER	Protocolo	Intercambio Clave	Autent.	Cifrado	Mac
DHE-RSA-AES256-SHA	SSLv3	DH	RSA	AES(256)	SHA1
DHE-DSS-AES256-SHA	SSLv3	DH	DSS	AES(256)	SHA1
AES256-SHA	SSLv3	RSA	RSA	AES(256)	SHA1
EDH-RSA-DES-CBC3-SHA	SSLv3	DH	RSA	3DES(168)	SHA1
EDH-DSS-DES-CBC3-SHA	SSLv3	DH	DSS	3DES(168)	SHA1
DES-CBC3-SHA	SSLv3	RSA	RSA	3DES(168)	SHA1
DES-CBC3-MD5	SSLv2	RSA	RSA	3DES(168)	MD5
DHE-RSA-AES128-SHA	SSLv3	DH	RSA	AES(128)	SHA1
DHE-DSS-AES128-SHA	SSLv3	DH	DSS	AES(128)	SHA1
AES128-SHA	SSLv3	RSA	RSA	AES(128)	SHA1
RC2-CBC-MD5	SSLv2	RSA	RSA	RC2(128)	MD5
RC4-SHA	SSLv3	RSA	RSA	RC4(128)	SHA1
RC4-MD5	SSLv3	RSA	RSA	RC4(128)	MD5
RC4-MD5	SSLv2	RSA	RSA	RC4(128)	MD5
EDH-RSA-DES-CBC-SHA	SSLv3	DH	RSA	DES(56)	SHA1
EDH-DSS-DES-CBC-SHA	SSLv3	DH	DSS	DES(56)	SHA1
DES-CBC-SHA	SSLv3	RSA	RSA	DES(56)	SHA1
DES-CBC-MD5	SSLv2	RSA	RSA	DES(56)	MD5
EXP-EDH-RSA-DES-CBC-SHA	SSLv3	DH(512)	RSA	DES(40)	SHA1
EXP-EDH-DSS-DES-CBC-SHA	SSLv3	DH(512)	DSS	DES(40)	SHA1
EXP-DES-CBC-SHA	SSLv3	RSA(512)	RSA	DES(40)	SHA1
EXP-RC2-CBC-MD5	SSLv3	RSA(512)	RSA	RC2(40)	MD5
EXP-RC4-MD5	SSLv3	RSA(512)	RSA	RC4(40)	MD5

6.2.2. Soporte en los navegadores instalados en el dispositivo Android

Para conocer el soporte de cifradores que disponen los navegadores instalados, el procedimiento no es tan directo. Se ha de editar manualmente la directiva SSLCipherSuite del archivo de configuración del Apache (`httpd.conf`) para cada una de las opciones posibles, listadas en la tabla 25, reiniciar el servidor, y solicitar el establecimiento de una conexión HTTPS (https://192.168.1.35) desde el dispositivo móvil. Si el cliente, en este caso el navegador, responde a las peticiones adecuadamente de forma que se visualiza la página web (almacenada en `c:/server/public/htdocs/index.html`), el navegador soporta dicho cifrador. Si no se consigue establecer la conexión, por ende el navegador no da soporte a ese cifrador.

Se muestra en la Tabla 26 el soporte a algoritmos de cifrado que ofrecen los navegadores nativo de Android, Firefox, Ninesky y Dolphin:

Tabla 26. Soporte cifradores navegadores Android

CIPHER	Nativo	FireFox	NineSky	Dolphin
DHE-RSA-AES256-SHA	✓	✓	✓	✓
DHE-DSS-AES256-SHA	×	×	×	×
AES256-SHA	✓	✓	✓	✓
EDH-RSA-DES-CBC3-SHA	✓	✓	✓	✓
EDH-DSS-DES-CBC3-SHA	×	×	×	×
DES-CBC3-SHA	✓	✓	✓	✓
DES-CBC3-MD5	×	×	×	×
DHE-RSA-AES128-SHA	✓	✓	✓	✓
DHE-DSS-AES128-SHA	×	×	×	×
AES128-SHA	✓	✓	✓	✓
RC2-CBC-MD5	×	×	×	×
RC4-SHA	✓	✓	✓	✓
RC4-MD5	✓	✓	✓	✓
EDH-RSA-DES-CBC-SHA	✓	×	✓	✓
EDH-DSS-DES-CBC-SHA	×	×	×	×
DES-CBC-SHA	✓	×	✓	✓
DES-CBC-MD5	×	×	×	×
EXP-EDH-RSA-DES-CBC-SHA	✓	×	✓	✓
EXP-EDH-DSS-DES-CBC-SHA	×	×	×	×
EXP-DES-CBC-SHA	✓	×	✓	✓
EXP-RC2-CBC-MD5	✓	×	✓	✓
EXP-RC4-MD5	✓	×	✓	✓

A partir de estos resultados, se deduce que los navegadores Ninesky y Dolphin soportan 14 cifradores de los 23 posibles, mientras que el nativo y Firefox solamente 8. El estándar DSS para autenticación no es soportado por ninguno de los dos navegadores. Firefox es incluso más limitado ya que tampoco da soporte al protocolo SSLv2, ni al algoritmo de cifrado simétrico DES. Ninesky soporta SSLv2 si el cifrador configurado es EXP-RC2-CBC-MD5, y sí soporta DES como cifrador simétrico.

6.3. Eficiencia

6.3.1. Estudio en tiempo de descarga

La medida de eficiencia en términos de coste temporal se ha realizado a través de una aplicación CGI (*Common Gateway Interface*), que se aloja en el servidor, (`c:\server\public\cgi-bin`) capaz de crear contenido dinámico para la medición del tiempo de descarga de ficheros de diferentes tamaños, desde 1KB a 1MB.

Es un programa que se ejecuta en tiempo real en el servidor web en respuesta a una solicitud realizada por el navegador del usuario. Por cada petición, el servidor ejecuta el

programa CGI pasándole los datos de esta solicitud, este programa escribe el HTML en la salida estándar y el servidor web la envía al cliente, de tal manera, que los CGI's permiten la generación dinámica de código HTML dentro de una propia página HTML. Dicho script fue desarrollado por David Maroto en su estudio de seguridad en dispositivos móviles Symbian. (Paredes 2008)

Las medidas se han repetido por lo menos veinte veces, para luego quedarnos con los diez mejores resultados que cumpliesen un intervalo de confianza del 95%.

Al ejecutarse estos scripts en el dispositivo móvil obliga a que el navegador utilizado soporte varias ventanas emergentes, por lo que en un principio no se ha podido ejecutar en el navegador nativo, ya que únicamente permite una única ventana emergente. Del resto de navegadores disponibles en la Play Store, sólo Mozilla Firefox Beta for mobile y NineSky 2.5.1 cumplían los requisitos necesarios.

Comparativa navegadores

A continuación se van a mostrar los resultados del estudio de eficiencia temporal solo para los cifradores comunes a ambos navegadores.

Para realizar una comparativa en términos de eficiencia temporal más completa entre los cifradores comunes a ambos navegadores, se han extraído los resultados obtenidos en función del tamaño de fichero para así visualizar de una forma más cómoda el estudio. No obstante, se incluirá un estudio más completo centrado en Ninesky, dado que soporta 14 de los 23 cifradores posibles.

Fichero 1KB

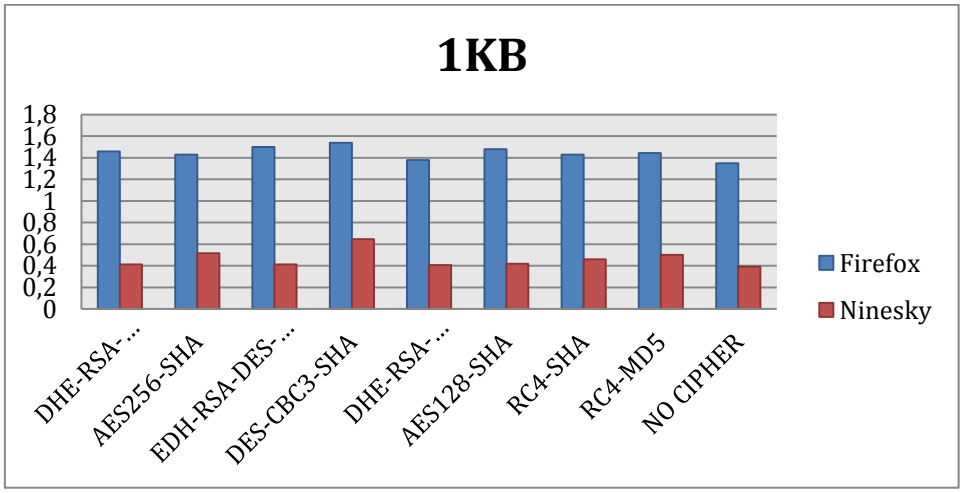


Figura 70. Resultados Navegador Firefox / Ninesky fichero 1KB

Como se observa a partir de la figura 70, los tiempos obtenidos con el navegador Ninesky son bastante inferiores a los obtenidos con Firefox. Este comportamiento se observará en todos los distintos tamaños de fichero cifrado. El tiempo promedio de

diferencia entre ambos navegadores alcanza un factor de 3.3, es decir, Firefox es más de tres veces más lento que Ninesky. En unidades temporales, el tiempo medio de descarga de un fichero de 1KB en Firefox ha sido de unos 1.4449 seg, y en Ninesky 0.4629 seg. En ambos casos el cifrador que ha alcanzado valores más lentos ha sido la combinación DES-CBC3-SHA, y DHE-RSA-AES128-SHA el que ha resultado ser el más rápido.

Midiendo la sobrecarga introducida por los diferentes cifradores, obtenida a partir de la diferencia del tiempo obtenido en cada cifrador con respecto a no usarlo, se han obtenido los siguientes resultados para un archivo de 1KB

Tabla 27. Resultados sobrecarga y eficiencia navegadores fichero 1KB

CIPHER	FIREFOX Sobrecarga (seg)	Eficiencia	NINESKY Sobrecarga (seg)	Eficiencia
DHE-RSA-AES256-SHA	0,1343	10,14%	0,0212	5,40%
AES256-SHA	0,105	7,93%	0,1234	31,46%
EDH-RSA-DES-CBC3-SHA	0,1759	13,28%	0,0193	4,92%
DES-CBC3-SHA	0,2131	16,09%	0,2533	64,57%
DHE-RSA-AES128-SHA	0,0548	4,14%	0,0152	3,87%
AES128-SHA	0,1559	11,77%	0,0262	6,68%
RC4-SHA	0,1046	7,90%	0,0675	17,21%
RC4-MD5	0,1183	8,93%	0,1091	27,81%

La sobrecarga introducida en términos porcentuales en Firefox se observa tiene un promedio de un 10% del tiempo. En cambio este factor aumenta considerablemente en el caso de utilizar Ninesky, cuyo valor más alto alcanza el cifrador DES-CBC3-SHA con un 64.57%, el cual ya se había destacado por ser el más lento. No obstante, la media de sobrecarga de este navegador es de 23.48%.

Fichero 10KB

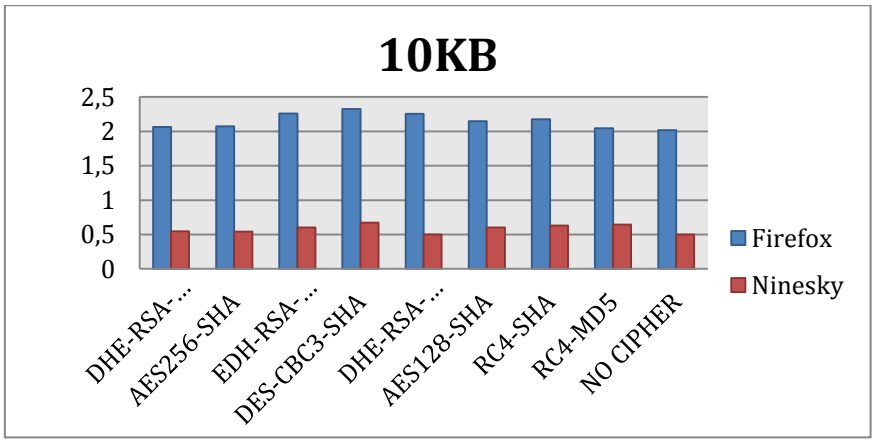


Figura 71. Resultados navegador Firefox / Ninesky fichero 10KB

Igual que en el caso anterior, y como ya se había anunciado, se observa como el tiempo en Firefox es considerablemente superior a Ninesky, este caso en un factor 3.7 en promedio, alcanzando la máxima diferencia con el algoritmo DHE-RSA-AES128-SHA. El tiempo promedio de cifra de un fichero de 10KB en Firefox es de unos 2.15 seg, y 0.62 seg para Ninesky.

El algoritmo más lento en ambos navegadores ha resultado ser DES-CBC3-SHA. En cambio, no ha coincido el más rápido, de forma que para Firefox ha resultado ser RC4-MD5 y para Ninesky DHE-RSA-AES128-SHA.

Los niveles de sobrecarga introducidos se muestran a continuación:

Tabla 28. Resultados eficiencia y sobrecarga navegadores 10KB

CIPHER	FIREFOX		NINESKY	
	Sobrecarga (seg)	Eficiencia	Sobrecarga (seg)	Eficiencia
DHE-RSA-AES256-SHA	0,0437	2,17%	0,047	9,43%
AES256-SHA	0,0537	2,66%	0,0417	8,37%
EDH-RSA-DES-CBC3-SHA	0,2388	11,83%	0,1029	20,65%
DES-CBC3-SHA	0,3053	15,13%	0,173	34,72%
DHE-RSA-AES128-SHA	0,237	11,74%	0,0025	0,50%
AES128-SHA	0,131	6,49%	0,104	20,87%
RC4-SHA	0,1556	7,71%	0,1337	26,83%
RC4-MD5	0,0241	1,19%	0,1454	29,18%

Se observa como la sobrecarga de Firefox está sobre un 7.37%, mientras que en Ninesky supera el 25%.

Fichero 100KB

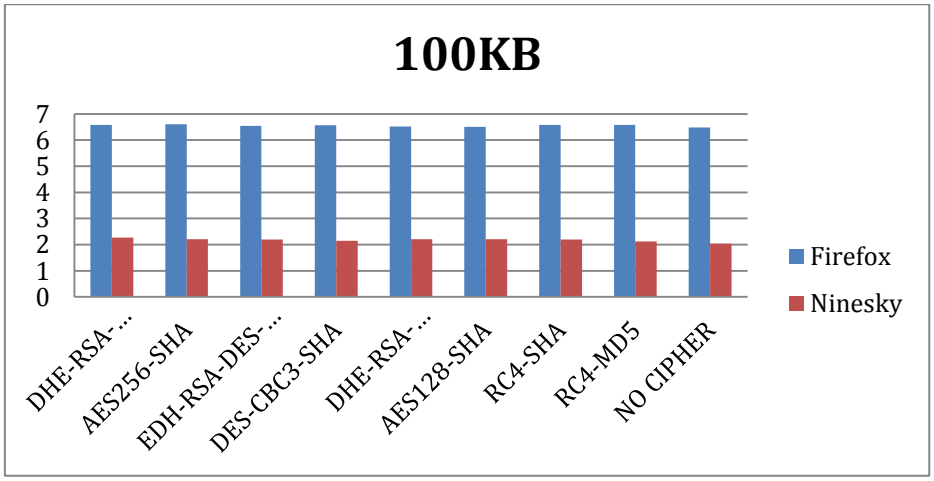


Figura 72. Resultados navegador Firefox / Ninesky fichero 100KB

Si el fichero es de tamaño 100KB la diferencia temporal entre ambos navegadores vuelve a ser de aproximadamente el triple. En valores temporales, el promedio en Firefox alcanza unos 6.55 seg y en Ninesky 2.27 seg. En este caso no coincide ni algoritmo más rápido ni el más lento. En Firefox es AES128-SHA el más rápido, no obstante, la diferencia con el resto de resultados es despreciable, diferenciándose en un 1.56% del algoritmo más lento, AES256-SHA. Esta diferencia es algo superior en Ninesky donde RC4-MD5 se despunta como el más rápido, aunque representa un 6.45% con respecto al algoritmo más lento, la combinación DHE-RSA-AES256-SHA.

Observando los términos de sobrecarga, se obtiene la siguiente tabla:

Tabla 29. Resultados eficiencia y sobrecarga navegadores fichero 100KB

CIPHER	FIREFOX Sobrecarga (seg)	Eficiencia	NINESKY Sobrecarga (seg)	Eficiencia
DHE-RSA-AES256-SHA	0,0986	1,52%	0,23	11,28%
AES256-SHA	0,1206	1,86%	0,162	7,94%
EDH-RSA-DES-CBC3-SHA	0,0555	0,86%	0,1569	7,69%
DES-CBC3-SHA	0,0869	1,34%	0,1087	5,33%
DHE-RSA-AES128-SHA	0,0333	0,51%	0,1673	8,20%
AES128-SHA	0,0178	0,27%	0,1732	8,49%
RC4-SHA	0,1011	1,56%	0,16	7,84%
RC4-MD5	0,1018	1,57%	0,0836	4,10%

Se observa como los valores de sobrecarga alcanzan valores muy pequeños, que ni siquiera supera un 2%, muy próximo por tanto al valor obtenido sin usar cifrador. El único valor que destaca sobre el resto es el del algoritmo que ha sido más lento para el navegador Ninesky, que ha obtenido un 11.28% de sobrecarga, cuando el resto de cifradores no superan un 9%.

Fichero 1MB

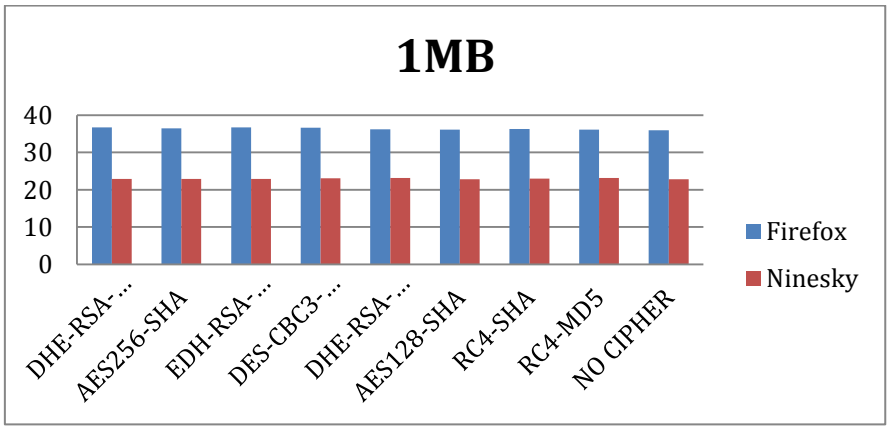


Figura 73. Resultados navegador Firefox / Ninesky fichero 1MB

Cuando el fichero es de gran tamaño, la diferencia entre navegadores no es tan notable, aunque si es perceptible que Ninesky sigue siendo más rápido aunque esta vez en un factor de 1.58. Esta información se visualiza de forma que en Firefox el promedio de cifrado es de 36.33 seg y en Ninesky 23.02 seg. En cambio, el algoritmo más rápido coincide para ambos navegadores, que corresponde a AES128-SHA. El más lento en Firefox ha resultado ser EDH-RSA-DES-CBC3-SHA, y en Ninesky RC4-MD5.

En cuanto a los valores obtenidos de sobrecarga, se muestran a continuación en la siguiente tabla:

Tabla 30. Resultados eficiencia y sobrecarga navegadores 1MB

	FIREFOX		NINESKY	
CIPHER	Sobrecarga(seg)	Eficiencia	Sobrecaa (seg)	Eficiencia
DHE-RSA-AES256-SHA	0,756	2,10%	0,1185	0,52%
AES256-SHA	0,4883	1,36%	0,1459	0,64%
EDH-RSA-DES-CBC3-SHA	0,7659	2,13%	0,1388	0,61%
DES-CBC3-SHA	0,6475	1,80%	0,2594	1,14%
DHE-RSA-AES128-SHA	0,2374	0,66%	0,393	1,73%
AES128-SHA	0,1123	0,31%	0,0486	0,21%
RC4-SHA	0,2969	0,83%	0,219	0,96%
RC4-MD5	0,1408	0,39%	0,3601	1,58%

Se observa como todos los valores toman valores muy bajos, promediados en un 1.2% en ambos casos.

6.3.2. Estudio eficiencia navegador más completo

A continuación se va a realizar un estudio centrado en el navegador Ninesky ya que admite cifradores adicionales a los soportados por Firefox.

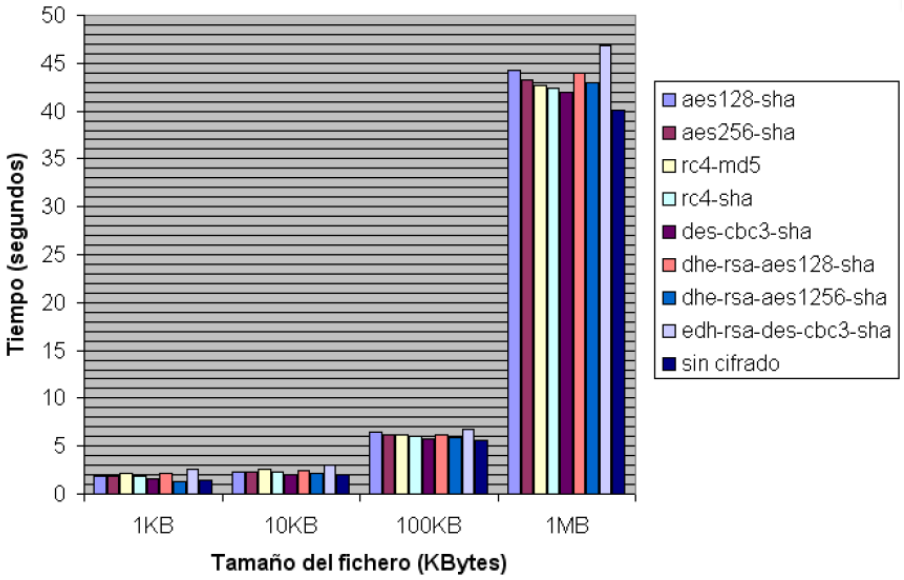


Figura 74. Resultados estudio completo Ninesky

Fichero 1KB

A partir de los resultados obtenidos se observa como el algoritmo AES128-SHA es la combinación más rápida, con resultados muy próximos a DHE-RSA-AES128-SHA, DES-CBC3-SHA, RC4-SHA Y AES256-SHA, mientras que los valores con mayor amplitud se han registrado en dos de los cifradores exportados, EXP-RC2-CBC-MD5 y EXP-RC4-MD5, junto con EDH-RSA-DES-CBC3-SHA. La diferencia temporal entre usar uno u otro es superior al doble.

Fichero 10KB

A partir del gráfico se observa como el algoritmo más rápido alcanza un valor muy próximo a no usar ningún algoritmo de cifrado, 0,5 segundos, y que corresponde a DHE-RSA-AES128-SHA. Los valores más grandes se alcanzan con el cifrador EXP-RC4-MD5 y con EDH-RSA-DES-CBC-SHA, superando los 0.73 seg, lo que supone un factor de 1.5 veces superior con respecto al más rápido.

Fichero 100MB

Destaca sobre todos los demás cifradores el más lento, EXP-RC2-CBC-MD5, que alcanza un valor de 2.98 seg, aunque la mayoría de ellos se sitúan por debajo de los 2.21 segundos, los algoritmos más rápidos son RC4-MD5 y DES-CBC3-SHA, con tiempos de 2.12 Y 2.15 segundos.

Fichero 1MB

El cifrador que tiene mejores resultados en un fichero de 1MB es AES128-SHA, con 22.81 seg, y los cifradores más lentos son EXP-RC2-CBC-MD5 y EXP-RC4-MD5, con 23.45 seg y 23.32 seg respectivamente. No obstante, el factor diferencia entre el más rápido y el más lento apenas supera 1.028, por lo que esta diferencia es despreciable.

6.4. Resumen y discusión

Este estudio ha servido para medir la eficiencia cuando se cifran las conexiones SSL. Para ello se ha estudiado en un primer lugar, cuales son los cifradores soportados por el navegador/es del dispositivo móvil Android. En este sentido se ha demostrado que el abanico de posibilidades de cifrado se ve bastante reducido ya que de los 23 posibles, en el mejor de los casos, Ninesky, soporta 14, y el número se reduce a 8 para Firefox.

El resumen de lo observado hasta este momento se puede ver en la siguiente tabla:

Tabla 31. Resumen resultados obtenidos

		1KB	10KB	100KB	1MB
Lento	Firefox	DES-CBC3-SHA	DES-CBC3-SHA	AES256-SHA	EDH-RSA-DES-CBC3-SHA
	Ninesky	DES-CBC3-SHA	DES-CBC3-SHA	DHE-RSA-AES256-SHA	DHE-RSA-AES128-SHA
Rápido	Firefox	DHE-RSA-AES128-SHA	RC4-MD5	AES128-SHA	AES128-SHA
	Ninesky	DHE-RSA-AES128-SHA	DHE-RSA-AES128-SHA	RC4-MD5	AES128-SHA

Basándonos en los resultados de este estudio, se puede concluir que el más rápido es el DHE-RSA-AES128-SHA, para un tamaño de archivo pequeño, y AES128-SHA si el tamaño del fichero es más grande. La combinación más lenta de algoritmos para un fichero pequeño es DES-CBC3-SHA y para un archivo más grande no hay una unanimidad, aunque las diferencias son muy escasas.

Capítulo 7

Conclusiones y trabajos futuros

7.1. Conclusiones

Se hace evidente que cada día las comunicaciones digitales son más complejas, y cada vez son más los escenarios en los que la seguridad de la información cobra una mayor importancia.

De forma habitual se comparten datos privados depositando la confianza en sistemas en los que un fallo de seguridad puede acarrear graves consecuencias. Por este motivo, la inversión que las empresas realizan para conseguir unas infraestructuras fiables es cada vez mayor.

En este estudio se ha pretendido dar una visión del estado de seguridad desde un punto de vista de dispositivos móviles Android. Se han estudiado las diferentes posibilidades existentes para almacenar información de forma privada mediante el cifrado simétrico y asimétrico en el propio dispositivo, así como la velocidad y seguridad de los diferentes algoritmos.

Se ha estudiado las diferentes opciones de seguridad disponibles para el establecimiento de conexiones a Internet desde el dispositivo móvil, a través de los diferentes navegadores.

7.2. Líneas futuras

El alcance de este proyecto ha sido el estudio de eficiencia de protocolos de medida de seguridad de Android, que podría ser completado o continuado con las siguientes propuestas:

- Estudio de eficiencia de consumo de batería

Este estudio se ha centrado en el consumo temporal, pero se podría complementar con un estudio energético, para ver qué operaciones y algoritmos son más costosos energéticamente hablando. Analizando ambos factores, se puede concluir qué algoritmo puede resultar más adecuado para una aplicación real.

- Análisis del API criptográfico Bouncy Castle

Aunque el alcance de este proyecto solo estaba fijado el análisis de la librería criptográfica que trae Android por defecto, durante la fase de investigación se observó que con la implementación de la librería BC, se ampliaba el abanico de posibles algoritmos de cifrado que podían llegar a ser estudiados (algoritmos asimétricos como ECDSA, ECC o funciones hash MD5 y SHA). Por ello, podría resultar interesante

realizar otra línea de investigación para obtener resultados más completos que abarquen un mayor número de algoritmos.

- Estudio soporte certificados

Estudio de los distintos tipos de certificados soportados por Android. Podría resultar interesante realizar un estudio sobre qué características debe cumplir un certificado X.509, X.968 y / o WTLS para que sea aceptado por el sistema operativo móvil.

- Uso de parámetros biométricos como semilla para métodos de cifrado

Análisis de eficiencia de los diferentes parámetros biométricos (huella dactilar, cadencia de escritura, imagen del rostro, voz…) que pueden ser utilizados como semilla para el cifrado. Se pueden analizar los patrones, como claves criptográficas, y analizar su robustez.

- Conciliación de los sistemas de cifrado con los requerimientos de intercepción legal

La legislación de algunos países obliga a facilitar la llave de descifrado de los propietarios cuyo contenido requisado se ha considerado sensible u/o que puede representar una amenaza nacional para utilizarlo como prueba ante un tribunal de justicia. Una posible línea de investigación es el estudio de la compatibilidad de algoritmos de cifrado con el requerimiento de dichos gobiernos para tener acceso al contenido requisado.

Capítulo 8
Presupuesto

A continuación se presenta el presupuesto asociado al desarrollo del presente proyecto.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor: Elma de la Fuente

2.- Departamento: Ingeniería Telemática

3.- Descripción del Proyecto:

- Titulo
- Duración (meses) **10**
Tasa de costes Indirectos: **20%**

4.- Presupuesto total del Proyecto (valores en Euros):

Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a titulo informativo)	Categoría	Dedicación (hombres mes) [a]	Coste hombre mes	Coste (Euro)	Firma de conformidad
					0,00	
Almenárez Mendoza, Florina		Ingeniero Senior	2	4.289,54	8.579,08	
Arias, Patricia		Ingeniero Senior	2	2.694,39	5.388,78	
de la Fuente, Elma		Ingeniero	10	2.694,39	26.943,90	
					0,00	
					0,00	
		Hombres mes	**14**	**Total**	40.911,76	

[a] 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable [d]
Ordenador portátil	900,00	100	9	60	135,00
Router inhalámbrico XAVi 7968	80,00	100	2	60	2,67
Tablet ViewPad 7ViewSonic	50,00	100	5	60	4,17
		100		60	0,00
		100		60	0,00
					0,00
				Total	141,83

[d] Fórmula de cálculo de la Amortización:

$$\frac{A}{B} \text{xCxD}$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO[e]

Descripción	Empresa	Costes imputable
Conexión a Internet		300,00
	Total	300,00

[e] Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y

6.- Resumen de costes

Presupuesto Costes Totales	**Presupuesto Costes Totales**
Personal	40.912
Amortización	142
Subcontratación de tareas	0
Costes de funcionamiento	300
Costes Indirectos	8.271
Total	49.624

Acrónimos

3GPP	3rd Generation Partnership Project
AC	Autoridad de Certificación
AES	Advanced Encryptation Standard
ANSI	American National Standards Institute
API	Application Programming Interface
CA	Autoridad de certificación
CBC	Cipher Block Chaining
CFB	Cipher Feed Back
CRHF	Collision Resistant Hash Functions
CRL	Certificate Revocation List
CSR	Certificate Signing Request
DES	Data Encryption Standard
DH	Diffie Hellman
DMS	Defense Messaging System
DSA	Digital Signature Algorithm
DSS	Digital Signature Standard
ECB	Electronic Code Book
ECC	Elliptic Curves Cryptography
FIPS	Federal Information Processing Standards
GSM	Global System for Mobile
IDEA	International Data Encryptation Algorithm
IP	Internet Protocol
ITU	Unión Internacional de Comunicaciones
JCA	Java Cryptography Architecture
JCE	Java Cryptography Extension
JDK	Java Development Kit
KSA	Key Scheduling Algorithm
LFSR	Linear Feedback Shift Register
MAC	Message Autentification Codes
MD	Message Digest
MDC	Modification Detection Codes
MITM	Men In The Middle
NBS	National Bureaux of Standards
NIST	National Institute of Standards and Technology
NSA	National Security Agency/Administration
OFB	Output Feed Back
OHA	Open Handset Alliance
OWHF	One Way Hash Functions

PFE	Problema de Factorización Entera
PGP	Pretty Good Privacy
PKG	Private Key Generator
PKI	Public Key Infraestructure
PKIs	Public Key Infraestructures (o ICP)
PLDE	Problema del Logaritmo Discreto Elíptico
PRGA	Pseudo-Random Generation Algorithm
PTH	Pseudo Transformaciones de Hadamard
RA	Autoridad de Registro
RFC	Request For Comments
RIPE	European RACE Integrity Primitives Evaluation
RIPEMD	RACE Integrity Primitives Evaluation Message Digest
SAFER	Secure and Fast Encryption Routine
SDK	Software Developmente Kit
SHA	Secure Hash Algorithm
SMIME	Secure Multipurpose Internet Mail Extensions
SSL	Secure Socket Layer
TCP	Transport Control Protocol
TLS	Transport Layer Security
TSA	Autoridades de Sellado de Tiempo
URI	Unifiorm Resource Identifiers
VA	Autoridades de Validación
WEP	Web Equivalent Privacy

Bibliografía

Adams, C. «The Cast-128 Encryption Algorithm.» Entrust Technologies, 1997.

Aguirre, Dr. Jorge Ramió. «Seguridad informática y criptografía.» Universidad Politécnica de Madrid, 2006.

Álvarez, Julio César García. «Curso de comunicaciones.» Ingenieria y arquitectura, Universidad Nacional de Colombia. (último acceso: 09 de 2013).

Android. *Dashboards | Android Developers.* http://developer.android.com/about/dashboards/index.html (último acceso: 09 de 2013).

Android Developers. *Android Security Overview.* http://source.android.com/devices/tech/security/index.html#android-platform-security-architecture (último acceso: 09 de 2013).

Android. *Security Tips | Android Developers.* https://developer.android.com/training/articles/security-tips.html (último acceso: 09 de 2013).

Blowfish. http://schneier.com/blowfish.html (último acceso: 09 de 2013).

Certicom Research. «SEC 1: Elliptic Curve Cryptography.» 2000.

FIPS. «Digital Signature Standard, DSS.» Computer Security, Cryptography, NIST, CSL, 1994.

Hernández, Fernando López. «Seguridad, criptografía y comercio electrónico con Java.» 2007.

Hernández, Fernando López. *Seguridad, criptografía y comercio electrónico con Java.* DELITmacprogramadores.org, 2007.

Lucena López, Manuel José. «Criptografía y seguridad en computadores.» Universidad de Jaén, 2011.

L, Guerrero, Gutiérrez E, y Murrieta E. «Ataque práctico al algoritmo RC4.» UNAM. (último acceso: 09 de 2013).

NIST. «Advanced Encryption Standard, AES.» Computer Security Standard, Cryptography, 2001.

NIST. «Data Encryption Standard (DES).» Computer Security, Cryptography, US. Department of Commerce, 1999.

NIST. «Internet X.509 Public Key Infraestructure Certificate and Certificate Revocation List (CRL) Profile.» 2008.

NIST. «SKYPJACK and KEA Algorithm Specifications.» 1998.

Menezes, A., P. Oorschot, y S. Vanstrone. *Handbook of applied cryptography.* CRC Press, 2011.

MINETUR, Inteco, OSI. *zonatic.usatudni.es.* http://zonatic.usatudni.es/aprendizaje/aprende-sobre-el-dnie/57-aspectos-tecnicos/196-criptografia-y-esquemas-de-clave-publica.html (último acceso: 09 de 2013).

Miret Biosca, Josep M. «Criptografía con curvas elípticas.» Matemàtica, EPS, Universidad de Lleida. (último acceso: 09 de 2013).

Oracle. *JCA Reference Guide.* http://docs.oracle.com/javase/6/docs/technotes/guides/security/crypto/CryptoSpec.html (último acceso: 09 de 2013).

Quirke, Jeremy. *Security in the GSM system.* AusMobile, 2004.

Paredes, David Maroto. *Estudio sobre seguridad en dispositivos móviles con sistema operativo Symbian.* PFC, Ingeniería Telemática, UC3M, Madrid: UC3M, 2008.

Sánchez, Rafael Ignacio Álvarez. «Aplicaciones de las matrices por bloques a los criptosistemas de cifrado de flujo.» Ciencia de la computación e inteligencia artificial, Universidad de Alicante, 2005.

Stallings, William. *Fundamentos de seguridad en redes: aplicaciones y estándares.* Pearson Education, 2003.

R.Rivest. *The MD5 Message-Digest Algorithm.* MIT and RSA Data Security, Inc, RSA, 1992.

RAE. *Real Academia de la Lengua Española.* http://lema.rae.es/drae/?val=criptograf%C3%ADa (último acceso: 09 de 2013).

Rivest, R. «A Description of the RC2(r) Encryption Algorithm .» MIT, RSA Data Security, 1998.

Rivest, R. «The RC5 Encryption Algorithm.» MIT, 1997.

RSA LAboratories. «RSA Cryptography Standard.» 2002.

Talens-Oliag, Sergio. *Seguridad en Java.* Instituto Tecnológico de Informática. 1999. http://www.uv.es/~sto/cursos/seguridad.java/html/sjava-40.html (último acceso: 09 de 2013).

Ternero, Mari Carmen Romero. «Seguridad en redes y protocolos asociados.» *http://www.dte.us.es.* 2003. http://www.dte.us.es/personal/mcromero/docs/ip/tema-seguridad-IP.pdf (último acceso: 09 de 2013).

Tovar, Rafael Francisco Benedicto. «Aplicación de metodologías de paralelización para la generación de tablas rainbow mediante la utilización de servidores de altas prestaciones en GNU/Linux.» EPS. Informática, Universidad de Almería, 2010.